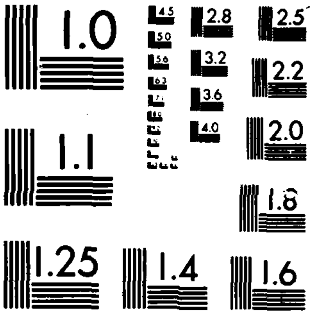
AD-A164 109 A MICROCOMPUTER-BASED MENU-DRIVEN CHIP LAYOUT LANGUAGE 1/4
(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH
SCHOOL OF ENGINEERING S C MORRESE DEC 85
UNCLASSIFIED AFIT/GCS/ENG/85D-11 F/G 9/5 NL



MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS-1963-A

1

A MICROCOMPUTER-BASED MENU-DRIVEN CHIP LAYOUT LANGUAGE

THESIS

Steven C. Morrese
Captain, USAF

AFIT/GCS/ENG/85D-11

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

1

AFIT/GCS/ENG/85D-11

A MICROCOMPUTER-BASED
MENU-DRIVEN CHIP LAYOUT LANGUAGE

THESIS

Steven C. Morrese
Captain, USAF

AFIT/GCS/ENG/85D-11

Approved for public release; distribution unlimited

AFIT/GCS/ENG/85D-11

A MICROCOMPUTER-BASED
MENU-DRIVEN CHIP LAYOUT LANGUAGE

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Computer Science

Steven C. Morrese, B.S.
Captain, USAF

December 1985

Approved for public release; distribution unlimited

PREFACE

This thesis was primarily concerned with the design and implementation of a Computer Aided Design (CAD) tool for use in the representation of the physical layout of Very Large Scale Integrated Circuits in the Caltech Intermediate Form on a microcomputer using the CP/M operating system (64K maximum memory). It provided me with the experience of working on a large software project and showed me the rate at which programs can grow in size while adding only simple functions.

MDCLL is designed to operate very similar to the chip layout language CLL, developed by Tim Saxe (15). In fact, all statements in the MDCLL system are taken from CLL. The only difference between CLL and MDCLL is that MDCLL uses a menu to obtain the information for representing the cell layout and doesn't implement all the functions of CLL (this is one of the downfalls of MDCLL and should be corrected in the future).

Thanks are offered to Lt Col Harold Carter, my thesis advisor, for persuading me to accept this challenging topic and seeing me through its development. Also, thanks go to my wife Sandra for putting up with me while I worked on this project.

Steven C. Morrese

Table of Contents

Appendices

List of Figures

List of Tables

ABSTRACT

The Menu Driven Chip Layout Language (MDCLL: pronounced Doctor Chill) is a microcomputer-based design tool used for representing the physical layout of integrated circuit designs in the Caltech Intermediate Form (CIF). MDCLL is based on the Chip Layout Language (CLL) created by Tim Saxe of Stanford University and as modified at the Air Force Institute of Technology. MDCLL leads the user through interactive menu driven functions, allowing the user to represent a circuit design in CIF; however, the user sees a language similar to CLL even though the file is stored on disk as CIF. The output CIF file is suitable for use with other Computer Aided Design (CAD) tools (e.g. MCIFPLOT) or for use in the fabrication process.

MDCLL uses a cell-oriented design methodology in which the user defines cells, and then the user combines these cells with the necessary interconnects to form the layout of an entire integrated circuit. Cell interconnections are presently possible by placing wires, abutting cells, or by overlapping two cells. In future versions the cell overlap method should produce a warning to the user if the cells overlap on more than just their predefined port interfaces. Another future enhancement would be to allow the interconnection of cells by the use of common naming of predefined ports.

The main functions of MDCLL are to create a new cell, delete a cell, place a cell, move a cell, modify a cell, interconnect cells, and print the CIF file of a cell.

1. Introduction

1.1 Background.

This thesis deals with two related items: Very Large Scale Integrated (VLSI) circuits; and microcomputers. In the following, a brief overview of integrated circuits and their design is presented followed by a look into the history of microcomputers.

1.1.1 Integrated Circuits.

In 1961, Fairchild introduced the first integrated circuit (IC). This revolutionary item contained a total of four devices (i.e. transistors) (7). Since that time, the number of devices per chip has grown astronomically to where presently there can easily be hundreds of thousands of devices on a single chip (3). The first IC's were designed by hand; the design was then transferred to rubylith, and finally the design was checked for accuracy by hand. The rubylith pattern was then optically reduced to form photolithographic masks. The complexity of the integrated circuits, however, was soon to cause the engineers problems in designing and testing. Due to the complexity involved, in the late 1960's, IC design and fabrication aids began to be used. The first assistance for the fabrication process to appear was the numerically controlled optical pattern generating machine. This machine required digitally encoded geometric patterns and the transfer of the layouts to data tape by use of electromechanical digitizers. This process encouraged the use of design rule checking programs to detect short circuits and spacing errors (4). During this period of time, there was no coordinated effort taking place to standardize the design and fabrication process. Each manufacturing facility was only concerned with their own specific problems and each tried solving the problems by themselves (each no doubt trying to get the competitive edge over the other).

A gap soon formed between academic engineering curriculums and the needs of the industry. This was caused by advancing technology in integrated circuits. Universities soon saw the need to not only provide IC design courses in their curriculums, but also the need to perform research in this rapidly growing technology. Carver Mead and Lynn Conway were pioneers in this effort. At the California Institute of Technology, in the early 1970's, they developed a course for not only studying IC design methodology, but also for laying out and testing actual circuits. In 1978 they produced the first book for use in the classroom environment. It is entitled Introduction to VLSI Systems, and thus the formal teaching of IC design methodology at the university level was started. (7 and 12)

1.1.2 Microcomputers.

Microcomputers were first built in the mid 1960's by engineering hobbyists (6). However, it wasn't until 1974 that the first microcomputer became available commercially when the Scelbi-8H was produced. The response was large and immediate, and the two people originally manufacturing this machine had to soon hire others. The next computer to become available was the MITS Altair which was featured on the front cover of the January 1975 issue of Popular Electronics (13). The next important event to occur was the introduction of the first integrated microcomputer in mid 1975. It was called the Sphere and contained the processor, keyboard, and display all in one case (1). In the same year (1975) many companies came out with microcomputers.

During the next ten years many companies tried to compete in the microcomputer market, with few surviving the competition. In 1980, a major breakthrough in the computer industry occurred when Sinclair (later bought out by Timex) marketed its ZX-80, the first computer sold for under $200. In 1981 another major innovation occurred when Osborne introduced the first transportable computer. IBM, seeing this new market for computers, in 1981 entered the microcomputer world with their IBM-PC. Not to be left behind, DEC entered the field in 1982 with the Rainbow 100 and Professional 325 and 350. In 1983, Osborne Computer filed for bankruptcy and, in 1984, Timex/Sinclair and others left the market or sold out. (2) The microcomputer business has not yet stabilized and probably won't for a few more years.

There are two reasons for the rapid acceptance of the microcomputer. One is the low cost of these new personal computers, and the other is the capability and flexibility of most of these machines. Microcomputers can do such varied tasks as keeping track of finances to controlling heating and lighting. Due to this capability and flexibility, microcomputers are being tried in as many different applications as possible. One of these applications is Computer Aided Design; the topic of this thesis.

1.2 Problem.

There currently exists a number of Computer Aided Design (CAD) tools available for the purpose of designing Very Large Scale Integrated (VLSI) circuits. Most of these tools are hosted on mainframe computers (e.g. Caesar) or dedicated minicomputers (e.g. the Apollo Domain workstation)(5). This makes the use of these tools very expensive. Due to this cost, many colleges and universities don't have any form of "hands-on" VLSI design courses available for their students.

To reduce the cost of VLSI design, it would be desirable to host a set of VLSI design tools on a microcomputer which would be capable of producing a file suitable for use by the chip manufacturing facilities. The difficulty with this is the lack of memory space and processing speed which would be available on a mainframe or dedicated minicomputer. The first problem to overcome is designing and implementing a tool for the physical layout of VLSI circuits on a microcomputer (the specific topic of this thesis). Afterwards, design rule checkers (for ensuring the layout is physically correct) and extractors (to use with logic simulation programs to ensure the layout will function as expected) are the next tools to be implemented on a microcomputer.

1.3 Summary of Current Work.

Until recently, all Computer Aided Design (CAD) was performed on mainframe computers or dedicated minicomputers. CAD tools are available on these machines due to their large memory capacity and fast processing speed. Since microcomputers are becoming both more powerful in terms of memory and speed and more prevalent, not only in the home, but also in business and educational institutions, CAD programs are being developed for the microcomputer. There are two sources of CAD tools available. CAD tools available in the private domain (i.e. commercially available) and CAD tools available in the public domain (mostly in educational institutions).

1.3.1 Private Domain.

Presently, most prevalent in the CAD area of tools for the microcomputer are automated graphics packages such as AutoCAD by Autodesk Incorporated (Mill Valley, CA) or 3D Space Tablet by Micro-Control Systems (Vernon, CT). These are powerful graphics tools which can draw practically anything (9). More in line with the idea of circuit design is the program, Logicsim, by E/Z Associates (San Jose, CA) which assists in printed circuit board layout. This program includes a library of not only the basic 7400 series TTL gates, but also some CD 4000 series CMOS gates (14). Another tool in the area of circuit design is the STRIDES package by Futurenet Corporation (Canoga Park, CA). This is a structured, interactive system, for designing integrated circuits. This powerful system takes advantage of a Winchester disk drive (5 and 11). Two other very capable programs in the area of circuit design are MICRO-CAP, an analog circuit analysis program, and MICRO-LOGIC, a digital design and simulation system. Both of these are by Spectrum Software (Sunnyvale, CA) (18 and 19).

Unfortunately, the programs, which are presently commercially available, are for use only down to the gate level of design (i.e. NAND, NOR, etc.). For designing VLSI circuits a tool is required for describing the physical layout of the transistors forming the circuit. The general purpose graphics packages described above could be useful for drawing VLSI circuits. However, not being designed for this purpose, their output files would not be compatible with existing tools (e.g. design rule checkers), nor would the output files be usable in the integrated circuit manufacturing process.

With the number of businesses and educational institutions working with VLSI design, and with microcomputers becoming more commonplace, it is easy to envision several software companies marketing packages specifically for the design of VLSI circuits on microcomputers in the very near future.

1.3.2 Public Domain.

Surprisingly, little work is taking place in the educational community dealing with putting a set of VLSI design tools on a microcomputer. This is most likely due to the fact that most universities (e.g. University of California at Berkeley) which have the knowledge base needed to perform VLSI design and create VLSI design tools already have mainframe computers for their use and will continue to have abundant access to these computers in the future. Only four such efforts exist: 1. The effort here at the Air Force Institute of Technology of not only this thesis but also the work of Capt. Kirk Horton who designed and implemented a program to perform checkplots based on a file in the Caltech Intermediate Form (7). 2. A program to print geometric files on an inexpensive line printer was developed by S. Sussman-Fort at Stoney Brook University, NY (17). 3. Kent Smith of Illinois University first developed a system for under $2000 hardware which would connect to a minicomputer host system using a design methodology called Path Programmable Logic (16). More recently, he has tried implementing this type of system on a single microcomputer (7). 4. M. Israel and G. Noguez from a University in Paris have designed and implemented a VLSI CAD tool called EMILIE2 for use on a microcomputer with a 0.7 MIPS (million instructions per second) microprocessor, 128K bytes of main memory, and 2 one Mbyte floppy disks with 0.15 second access times. The EMILIE2 system includes a design rule checker, an extractor, a simulator, and a plot program (8). One possible problem with this system is transportability to other microcomputers since the microcomputer on which it is implemented has such high capabilities. The only other public domain tool available is by Tom Alamy of Tektronics Corporation. He is attempting to host a set of CAD tools on a TRS-80 (7).

1.4 Approach.

The goal of this thesis is to implement a menu driven form of the existing VLSI CAD tool CLL (15) on a CP/M (Control Program for Microcomputers) operating system based microcomputer. Only the basic function of CLL, which is to act as a compiler to the Caltech Intermediate Form (CIF) of describing circuits, will be implemented. The CIF produced by the Menu_Driven Chip Layout Language can then be used by production facilities to fabricate integrated circuits. The output CIF file must be compatible with the program MCIFPLOT (7) and with existing CAD tools (e.g. the BANE preprocessor hosted on the AFIT UNIX-VAX 11/780).

2. REQUIREMENTS

2.1 Overall.

The basic requirement of the Menu Driven Chip Layout Language (MDCLL) is to allow easy (a relative term) development of an integrated circuit layout on a microcomputer. It must be compatible with the Caltech Intermediate Form (CIF) Version 2.0 (12) to allow production of the circuit and the use of existing Computer Aided Design (CAD) tools and existing CIF files. Table 2.1 shows all the CIF commands, and indicates whether they will be required or optional (nice to have if time permits). Descriptions of geometric elements in non-Manhattan directions (directions that form other than ninety degree angles) will not be allowed. These requirements will ensure compatibility with CIF files created by the chip layout language CLL (15) and will also ensure compatibility with the program MCIFPLOT (7).

CIF Command	Required	Optional
Polygon with path		X
Box with length, width, center, and direction	X	
Round Flash	X	
Wire with width and path		X
Layer specification	X	
Start symbol definition	X	
Finish symbol definition	X	
Delete symbol definition		X
Call symbol	X	
Comments	X	
End marker	X	

TABLE 2.1 CIF commands to be implemented

2.2 Implementation.

To obtain the most widespread use possible, MDCLL must be designed to function on most microcomputers. To facilitate this the program will be designed using a CP/M operating system with 64K of main memory (48K Transient Program Area) and two 180K disk drives (average 5 1/4" single sided, double density floppy disk). It will be written in the C programming language as described in Kernighan and Ritchie (10) and as implemented by the C/80 compiler. To allow MDCLL to be executed on as many systems as possible any system and compiler unique features should be avoided. CP/M was chosen since it is one of the most widespread standardized operating systems for microcomputers. The C language was selected due to its hierarchical structure, standardization, compact assemblage, and ability to manipulate low level I/O functions.

2.3 System.

The MDCLL system must be capable of performing the following functions: create new cells in CIF; delete, place, or move an existing cell with relation to other cells; modify an existing cell file; interconnect cells; and print a CIF file or plot a circuit's floorplan. (These functions are described in the following paragraphs.) It must also be able to input cells from existing libraries and also be able to iterate any given cell. To save on main memory space, each one of the above mentioned functions will be implemented as a separate program stored on disk and called into main memory only if and when it is needed. Not only will the amount of main memory required for program execution be minimized, but also the execution speed thereby enhancing the humancomputer interface. Each menu must be totally self explanatory with instructions and detailed error messages showing the user exactly what needs to be entered. The responses to the menus will, except for numeric responses, be single letter entries.

2.3.1 Create Function.

MDCLL must be able to create a new CIF file based on the users answers to menu driven design questions. Desired functions to be performed are: name the new cell, define a new layer, place a rectangle, lay a wire, place a via, place a round flash, place a polygon, and insert a port for connection to another cell. Any of these items can be given a name (with optional point location and layer) for easy identification in the CIF file (and on the CIF plot using MCIFPLOT).

2.3.2 Delete, Place, or Move Function.

The MDCLL system must allow the user to delete a cell from the hierarchy, place a cell in a given location, or move cells from one location to another in order to create an overall integrated circuit. This function should warn the user if any cells overlap in areas other than the predefined ports. Also available in this routine will be the ability to iterate cells.

2.3.3 Modify Function.

MDCLL must allow the user to easily modify any cell in the circuit without affecting the other cells. This function will allow the user to modify not only the primitive elements (lowest level cells in the design hierarchy) but also any other cell in the hierarchy. To maintain a structured approach to designing an integrated circuit, only interconnections should be included with calls to other cells in any cell other than a primitive element. If the size of the cell changes the user will be notified so other cells in the circuit may be adjusted accordingly.

2.3.4 Interconnect Function.

The MDCLL system allows the user to interconnect cells via the cells predefined port names. Initially, only a manual mode of interconnection will be implemented by use of the CLL wire command format (15). If an interconnection is made to other than the predefined ports a warning message will be issued. An automatic interconnection routine would be desirable, but will not be implemented in this first version of MDCLL.

2.3.5 Print/Plot Function.

The MDCLL system should provide both a print and plot routine. The print routine should allow the user to print any CIF file either on the screen or to a printer. The plot routine should allow the user to plot any circuit's bounding box diagram to a printer or plotter or the entire CIF layout, using MCIFPLOT (7), to a plotter.

2.4 Performance.

The main objective associated with this project is to provide an easy to use system for describing the physical layout of VLSI circuits. The integrated circuit designer must be able to use MDCLL without becoming bored waiting for the computer to respond or frustrated with hard to use menus. This means the response time for the computer should be no more than a few seconds for any command with no command taking longer than expected without advising the user of the processing involved (i.e. the user knows it takes longer to save a larger file to disk than a smaller file). Descriptive inquiries and error messages must be provided to ensure the user does not become confused as to what the system needs as a response. Therefore, the menus must be explicit and the error messages must tell the user exactly what is needed as input.

Another primary concern is the size of integrated circuit the MDCLL program will support. Unfortunately, this can only be described in terms of memory size and not actual number of transistors on chip (or other more meaningful terms). The usable memory will be limited by the largest subprogram in the MDCLL system. However, since this is a cell-oriented design system, this is only the limit of one cell in the circuit. Any number of cells can be combined to form a circuit. There is no reason why a designer could not use an entire diskette for one circuit thereby making the upper limit the maximum size the diskette can store.

3. System Design

3.1 Methodology.

Since the Menu Driven Chip Layout Language (MDCLL) is an extremely interactive system the main concern during design is the ease of use for the VLSI designer. One factor involved in the ease of use is the response time of the computer. To aid in handling this problem, the system is segmented into eight separate programs with only one program being resident in main memory at any time and the other programs being called in as needed. This allows maximum memory space for use as a work and storage area for the programs currently running. Only after one specific cell is completed will the information be written out to disk. By doing this, the system advocates (and even somewhat forces) the use of small cells (e.g. a subtractor or master slave flip flop) which would then be linked together, with the necessary interconnects, to form larger cells and eventually entire circuits. This segmentation into cells serves to enhance the understandability of the circuit being designed. This is sometimes referred to as a highly modularized top-down design, bottom up implementation methodology for integrated circuit design. It is very similar to the Top Down Structured Programming technique used by software engineers to produce the most reliable and maintainable code possible. Integrated circuit designers will have to eventually adopt some form of top-down structured design approach due to the high density and complexity of present and future VLSI circuits in order to ensure a useful end product.

The following paragraphs describe the functional composition of the main program and each subprogram (also referred to as programs) in the MDCLL system. The programs in the system are MDCLL, CREATE, DPM (Delete, Place, or Move), MODIFY, INTER, PRIPLO, FINALIZE, and SAVECELL. Only the basic functions of each program are described here; for complete details of the implementation of each program see Chapter 4.

3.2 MDCLL Program.

The MDCLL executive program will control the execution of the other programs in the MDCLL system. It will perform several major functions. It will initialize the system, display the main menu for the user, and be responsible for bringing into main memory the requested program. The requested program will then be executed and control will return to the main menu upon completion. From the user's standpoint, the MDCLL system can be considered as a *main menu* calling other menus in the MDCLL system based on user selected options. This is functionally shown in Figure 3.1.

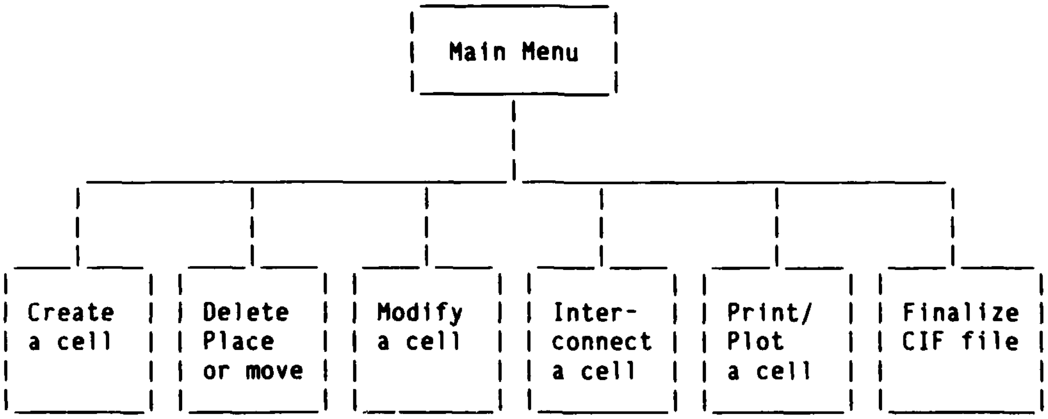


Figure 3.1 MDCLL User Functions

Figure 3.2 represents the functional layout of the MDCLL program, as described above, along with the subprograms which can be executed from the MDCLL program.

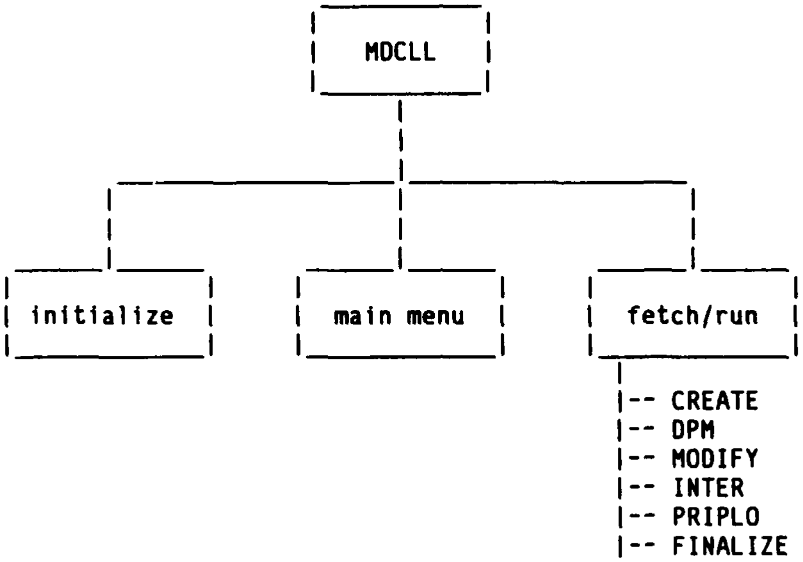


Figure 3.2 Executive Level Program

The initialization function must present the user with the revision number and date of the program being executed along with some basic instructions on using the system.

The display main menu routine must be capable of not only displaying the user's options in an easy to read, easy to understand format, but also prompt for a response, check the validity of the response, and, on non-valid entries, produce explicit error messages to the user to ensure a correct entry. The options to be displayed allow the user to create a cell (CREATE); delete, place, or move a cell (DPM); modify a cell (MODIFY); interconnect cells (INTER); print files or plot CIF layouts (PRIPLO); finalize a circuit (FINALIZE); or allow the user to exit the system. Each of these, except for exiting the system, will be a separate program called into memory when requested.

The fetch and run function must be capable of retrieving the correct program from disk and prompting the user to insert the proper disk if the program requested cannot be found on the current disk. This function must also be capable of then running the desired program. As shown in Figure 3.2 each of the subprograms can be considered as subordinate to this function.

3.3 CREATE Subprogram.

The CREATE subprogram of MDCLL must allow the user to create a new CIF file based on responses to menu driven questions. The cells, thus created, are considered to be the basic elements of the circuit being designed since all circuits are composed of these cells (termed primitive cells) grouped together with the appropriate interconnections. The CREATE subprogram is the most important since it will produce the primitive cells for use in later phases of the design process. The file created will be a totally CIF compatible file and can be used with other design tools (e.g. design rule checkers and MCIFPLOT). A separate file on disk will be created for each new primitive cell created and will be designated with a ".cif" extension. As Figure 3.3 shows, the main functions of CREATE are naming the cell, displaying the menu, and implementing the desired option.

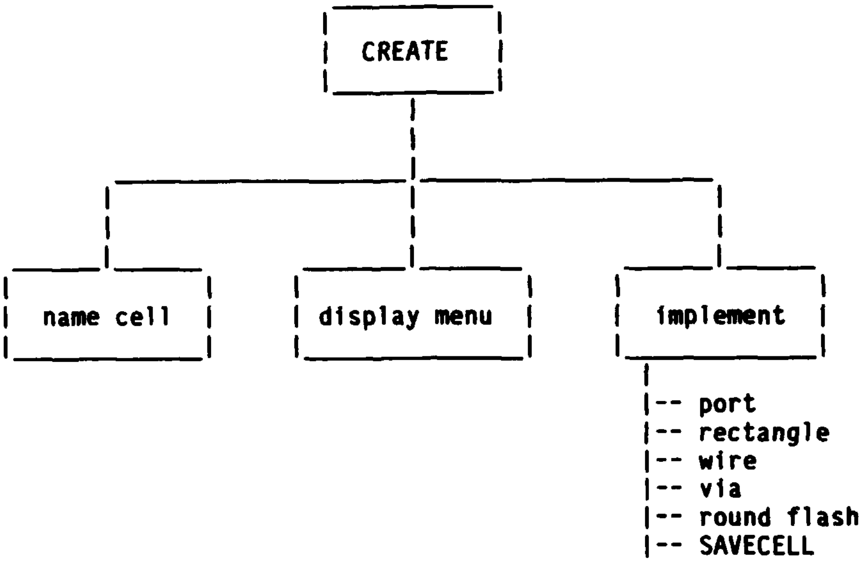


Figure 3.3 CREATE subprogram

The options performed in creating a primitive cell consist of placing an interconnection port, rectangle, wire, via, or round flash. Other options available will be to exit to the main menu after saving the cell or quit without saving the cell. Figure 3.3 shows the function of implementing an option will not only incorporate sub-functions to handle these capabilities, but also includes the execution of the SAVECELL subprogram.

3.4 DPM Subprogram.

The functions to delete, place, or move an existing cell (DPM subprogram) are grouped together since their functions are similar. These functions will make extensive use of the same subroutines. Figure 3.4 shows the functional chart for this subprogram. The three main

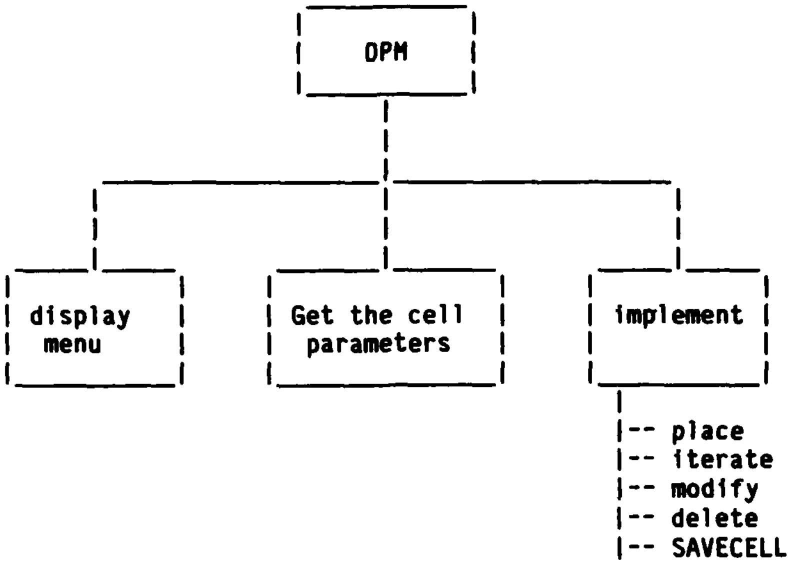


Figure 3.4 DPM subprogram

functions of DPM are to display the menu of possible options, fetch the cell parameters, and implement the desired function.

The delete function must allow the user to delete a cell from the design. This includes not only the primitive cells but cells anywhere in the hierarchy of the design along with iterated cells. The place function must allow for the creation of new levels of the hierarchy and place lower levels into the new level. These higher levels of the design (any level above the primitive cells) will also be written to disk and will also be totally CIF compatible and therefore designated with a ".cif" extension. (To obtain CIF files from these cells, usable in MCIFPLOT, the user must first enact the FINALIZE program under the main menu. This program is described in paragraph 3.8 below.) The place function must also have the capability of placing primitive cells from an existing library of cells. It must also have the capability to iterate cells as in the CLL language. The modify function must allow the user to move cells within the parent cell layout. It should also give warning messages if the movement causes any cells to overlap and allow the user to change the interconnections between these cells as needed. As Figure 3.3 shows, the function of implementing an option will not only incorporate sub-functions to handle these capabilities, but also includes the execution of the SAVECELL subprogram.

3.5 MODIFY Subprogram.

The MODIFY subprogram will allow the user to modify not only any primitive cell in the system but also any cell in the hierarchy of cells making up a circuit. As shown in Figure 3.5 the major functions needed to perform a modification are to retrieve the requested file, determine the type of modification (i.e. add, delete, or change parameters), display a menu of options available for modifying a cell, and then implement the desired option. If the type of modification to be performed is to add parameters to the cell, then the menu used for creating a cell along with the implement function for creating a cell are used (Paragraph 3.3). This is to reduce the complexity and number of modules in the MDCLL system.

The options which can be performed in modifying a cell, other than adding to the cell, consist of modifying or deleting an interconnection port, rectangle, wire, via, or round flash. Other options available will be to exit to the main program after saving the cell or quit without saving the cell. As Figure 3.5 shows, the function of implementing an option will not only incorporate sub-functions to handle these capabilities, but also include the execution of the SAVECELL subprogram.

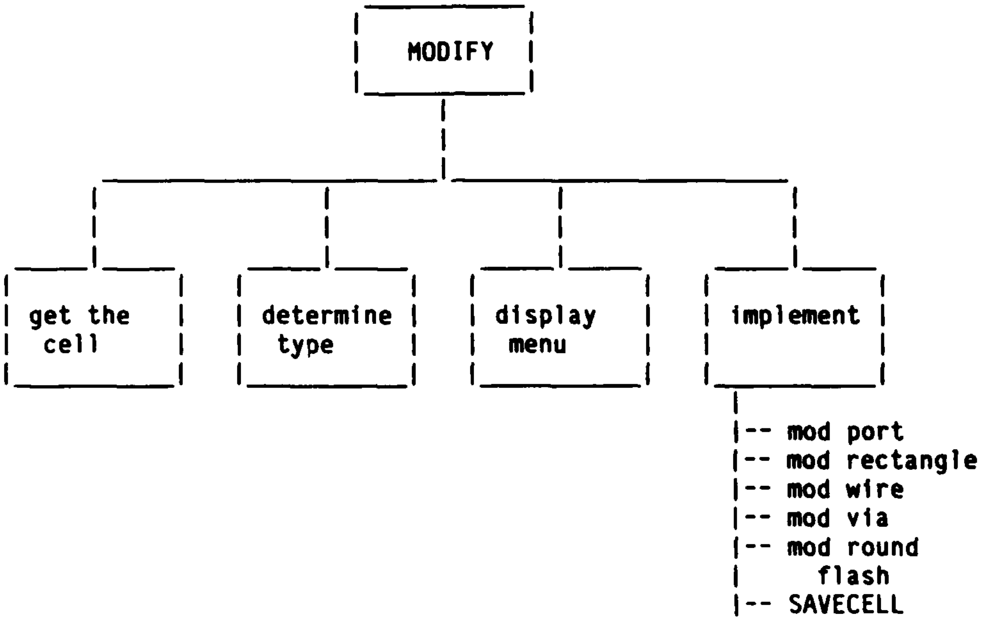


Figure 3.5 MODIFY subprogram

3.6 INTER Subprogram.

The INTER subprogram will allow the user to interconnect cells already placed with the place/move functions described in Paragraph 3.4. Figure 3.6 shows the main functions of this subprogram. The functions needed to be performed are to determine and get the parent cell and parameters for the cells to be interconnected, connect the cells with wire statements as defined by the user, and then save the cell using the subprogram SAVECELL.

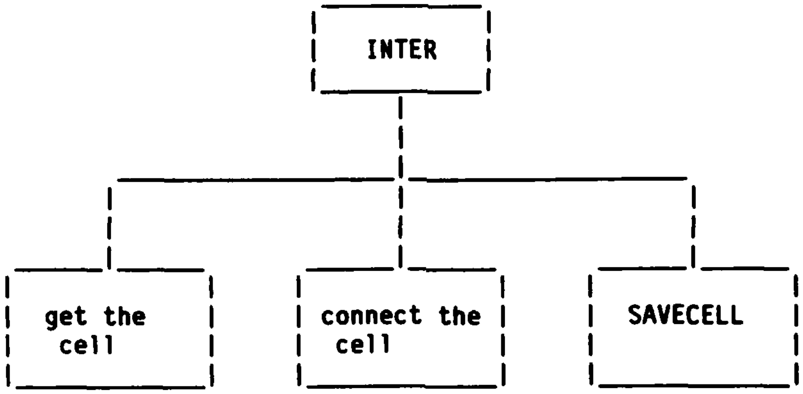


Figure 3.6 INTER subprogram

3.7 PRIPLO Subprogram.

The PRIPLO subprogram contains two highly related functions. The print function of this program should allow the user to print any ".cif" file on either the screen or printer. The plot function should allow the user to plot the CIF layout of the cell using MCIFPLOT on a plotter

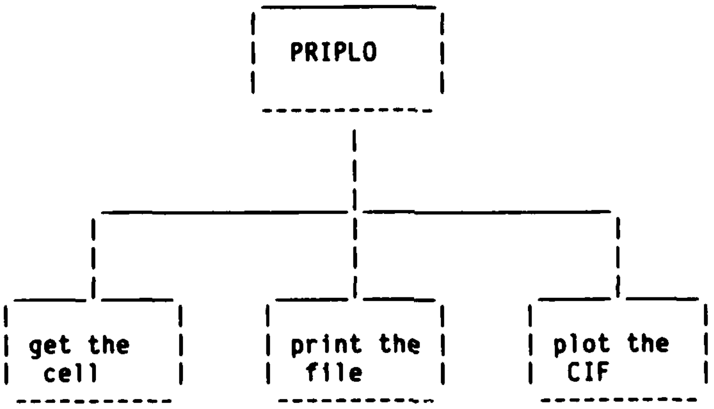


Figure 3.7 PRIPLO subprogram

or just the bounding box diagram on a printer or plotter. This will allow the user to visualize the entire circuit a section at a time. As shown in Figure 3.7 the main functions of this subprogram are to get the cell and perform the print or the plot.

3.8 FINALIZE Subprogram.

The FINALIZE subprogram should only be used when a circuit is finished and a complete CIF file is needed or when the user needs to use MCIFPLOT to obtain a plot of a cell which calls subordinate cells. The finalization routine will start at the highest level of the hierarchy for the circuit and create a ".cif" file based on the calls to other lower level cells and any interconnection wires contained therein. The main functions of this program are to retrieve the cells and then format the CIF file while processing the call statements. See Figure 3.8 for the functional chart showing this.

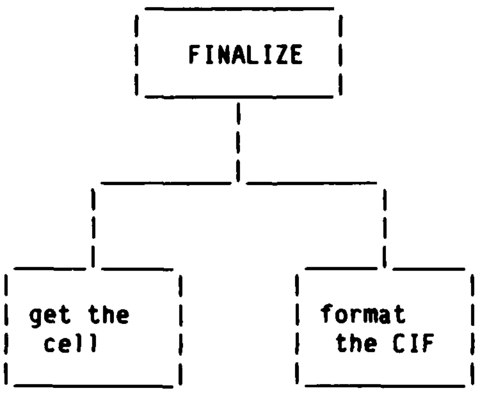


Figure 3.8 FINALIZE subprogram

3.9 SAVECELL Subprogram.

The SAVECELL subprogram is executed by the other subprograms in the MDCLL system to save a cell to disk. SAVECELL performs only one basic function which is to save the information stored in memory out to disk. However, this basic function can be broken down, as shown in Figure 3.9, into outputting the CIF file header, the CIF cell layout, and the CIF file footer.

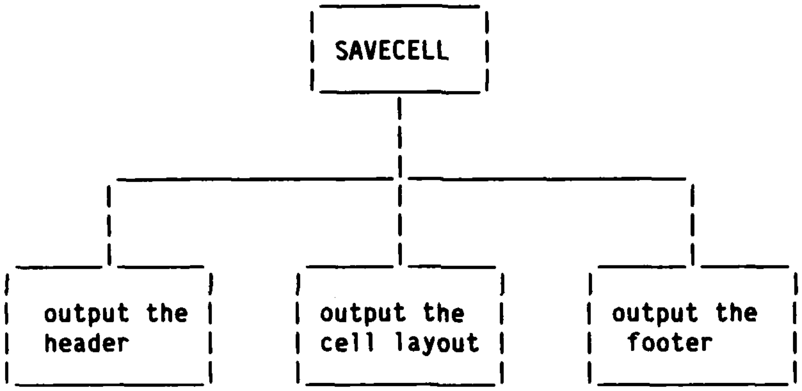


Figure 3.9 SAVECELL subprogram

4. Detailed Design

The programs of the MDCLL system are designed using small, independent modules. Each program (actually subprograms, but since each one can stand alone they are all referred to as programs) in the system not only has its own modules available to it but also the modules of any of the other programs through the use of libraries included in each program as needed. This helps to reduce the amount of code needed. Also, since there are no duplicate modules to contend with, the use of standard libraries aids in future maintainability.

The detailed design started with the upper level hierarchy charts shown in Chapter 3. These were then further broken down, one program and one module at a time. The final hierarchy charts for the system are shown in Appendix A along with the structure charts. The hierarchy charts are also included in the header to each program in the system (see Appendix B). Hopefully, this will encourage future maintainers to not only update the code but also the hierarchy charts. An English-like Program Design Language (PDL) was then used to give each module the final detailed design which could then be coded. This PDL is included in the header to each module, as shown in Appendix B, to once again allow the programs to be maintained more easily.

The following paragraphs describe in detail the design of the entire MDCLL system. First, the header used throughout the system is described. The main program (MDCLL) is then described followed by each of the subprograms which it is capable of calling (CREATE, DPM, MODIFY, INTER, PRIPLO, and FINALIZE). The SAVECELL subprogram is described last since it is called by the CREATE, DPM, and MODIFY subprograms.

4.1 Header.

The header was designed to be standard for each program in the MDCLL system. This standard header is used at the beginning of each program in the system even though not every program uses every part of the header. This is once again for ease of maintenance. There are four separate sections to the header: the include files needed, the global constants defined, the global variables, and the structures for storing the MDCLL Intermediate Form representation (MIF: not quite the Caltech Intermediate Form but almost).

4.1.1 Include Statements.

First in the header are two include statements. These provide the necessary code for the function of printing to the terminal (the "printf" command) and the function of executing one program from another (the "exec" command: a microcomputer version of the "system" command described in Kernighan and Ritchie). Each of these commands is used in each of the programs in the MDCLL system.

4.1.2 Global Constants.

Next in the header are the global constants. These are provided up front so when design rule changes are needed the constants associated with these rules will be easily accessible. These constants are for lambda to CIF scaling factors and via sizes. Also included here are the standard declarations needed by most programs: end of file, true, false, and null. The last constant defined is the amount of available memory space for the stored MIF. This is used in determining the memory space left in the system in comparison with the amount used when adding to the layout of cells. This is provided so the user doesn't try to exceed the memory capabilities of the machine, although in this version the user is not prevented from doing so. It is initially set at 20K which is derived in the following manner. Start with a 64K system; subtract the 16K needed for the CPM operating system; subtract the size of the largest program in the MDCLL system to ensure all programs can be run on any cell (26K for the MODIFY program). This leaves 22K usable memory. Leaving 2K as a buffer gives 20K or 20480 bytes of usable memory.

4.1.3 Global Variables.

When this project was started, no global variables (except the structures described next) were envisioned. As time became a factor and things were getting frantic (situation normal for software development) global variables suddenly appeared. These variables should be placed back into the modules as passed parameters as time permits.

4.1.4 Structures used for Storage.

Finally the most important part of the header: the structures for storing the MDCLL Intermediate Form (MIF) which will be translated to the Caltech Intermediate Form (CIF) upon output to disk. The seven separately defined structures are inter_ports, rectangle, via, wire, call_statement, iter_statement, and called_cells.

4.1.4.1 inter_ports Structure.

The inter_ports structure is used for holding the information describing an interconnection port. This is the portname, the layer in which the port resides, and the x,y location of the port. Also included in this structure is a pointer to the next port to form a linked list of interconnection ports.

4.1.4.2 rectangle Structure.

The rectangle structure is used not only for holding rectangle information described by the user, but also for holding the necessary information to describe user defined wire statements. This second purpose is only used when the wire is being saved to disk as a series of rectangles by the SAVECELL program. The information needed to describe a rectangle (or box) in CIF is the height, length, and x,y location. Also included in this structure are three pointers to the next defined rectangles. These pointers form a trinary tree of rectangle definitions based on the value of the x location. There are eight variables used as pointers to the rectangle structure. Four of these point to the initial rectangles defined by the user, one for each layer available. The other four point to the rectangles, by layer, needed for the conversion of wires`to CIF. This form of storage was chosen to allow the user to define and modify the rectangles by layer. It was also chosen so only one CIF layer statement would be generated for all the rectangles in a cell for a given layer and only one CIF layer statement for all the rectangles generated by wire statements in a given layer. This reduces the amount of CIF produced compared with the Chip Layout Language (CLL), which produces a new CIF layer statement each time it encounters a different layer instead of gathering rectangles by layer as MDCLL does.

4.1.4.3 via Structure.

The via structure is similar to the rectangle structure in that it is used for not only holding user defined via information but also for storing the necessary via information for user defined wire statements that change layers. These wire defined vias are only used by the SAVECELL program. The only information needed for a via is the x,y location and the layer as given by the pointer. As in the rectangle structure, the via structure has three pointers defined. These pointers form a trinary tree of via definitions based on the value of the x location. This form of storage was chosen to allow the user to define and modify the vias by layer. It was also chosen so only one CIF layer statement would be generated for each of the parts of the vias in a given layer and similarly only one CIF layer statement for each part of the vias generated by wire statements in a given layer. This reduces the amount of CIF produced compared with the Chip Layout Language (CLL), which produces a new CIF layer statement each time it encounters a different layer instead of gathering vias by layer as MDCLL does.

4.1.4.4 wire Structure.

The wire structure is different from the previously defined structures. This is caused by the fact that a wire has an unspecified number of segments. For this reason the wire structure contains only the starting x,y location, a pointer to the first set of wire pa/ameters (first wire segment), and the three pointers forming a trinary tree to other wires based on the value of the x location. There is only one trinary wire tree structure defining the starts for wire statements.

This single structure is used to allow the user to modify wire segments, at will, including not only the width and length, but also the segments' layers. The only reference is by x location. The wire is converted to rectangles just before saving the cell to disk which allows for savings in layer statements as described above for the rectangle structure. The internal structure wire_parameters contains the necessary information for each segment of the wire. This is the width, length, layer, and direction of the wire segment. Also included is a pointer to the next wire segment.

4.1.4.5 call statement Structure.

The call_statement structure holds all information needed to create a CIF call statement. This includes the name of the cell to be called, the associated CIF number, the location to where the cell is to be translated, if the cell is to be flipped upside down (mirrored in y in CIF language), if the cell is to be flipped left to right (mirrored in x), the rotational position of the x axis (limited as in CLL to the 3, 6, 9, and 12 o'clock positions), and the bounds of the cell being called. As in the wire structure, the call_statement structure has an internal structure associated with it. This is the named_ports structure which defines the names for interconnection ports for the called cell which should match in number to the interconnection ports defined in the called cell. Included in the named_ports structure is the port's name and a pointer to the next port. The call_statements are kept in a trinary tree based on the value of the CIF number.

4.1.4.6 iter_statement Structure.

The iter_statement structure contains the necessary information to iterate a cell. The information stored is the name of the cell to be iterated, the CIF number of the cell, the number of iterations in the x and y directions, the x and y pitch or spacing of the cells, and a pointer to the call_statement structure for holding the translation, the flipping instructions, the direction of the rotation, the bounds of the cell, and any associated port names. As with the call statements the iterate statements are stored as a trinary tree based on the value of the CIF number.

4.1.4.7 called cells Structure.

The called_cells structure is used by the FINALIZE program to ensure all the called cells are included once, and only once, in the final CIF file. It is a linked list containing the cellnames and CIF numbers of the called cells.

4.2 MDCLL Main (Module 0.0).

The MDCLL program is broken down into three basic functions, as shown in Figure 4.1 and the hierarchy chart in Appendix A. A single character is accepted as an input parameter. Based on this character the Initialization routine (Module 0.1) may or may not be executed. Then the options are displayed for the user (Main_Menu Module 0.2) and the proper program is finally executed (Fetch_Run Module 0.3).

```
MDCLL Main
  |
  |-- Initialize (Para 4.2.1)
  |
  |-- Main_Menu (Para 4.2.2)
  |
  |-- Fetch_Run (Para 4.2.3)
```

Figure 4.1 Modules for MDCLL Main

4.2.1 Initialization (Module 0.1).

The initialization function provides a brief message describing the program including the version being executed. It also provides the user with instructions not only on using the system, but also on how to not get the instructions in the future if not wanted.

4.2.2 Main Menu (Module 0.2).

The main menu function simply provides the user with the option of running one of the MDCLL programs or exiting to the operating system. If the selected character is not a valid option an error message stating so is printed along with the menu. The user is then asked to choose again. Once a valid option is obtained, this option is then passed back to the main module.

4.2.3 Fetch Run (Module 0.3).

The fetch and execute function accepts the option of which program is to be executed as input. This module then makes sure the requested program is on disk and then executes it. Due to this, the six programs

CREATE, DPM, MODIFY, INTER, PRIPLO, and FINALIZE should be considered subprograms to MDCLL although in this text they are referred to as programs.

4.3 CREATE Main (Module 1.0).

The CREATE program is used to create a new cell based on user inputs to menu driven questions. It has three basic functions. It first gets the cellname from the user (Module 1.1 Name_Cell) and determines if the name is already in use. If the name is already in use (on the current disk) then the name is not allowed and the user must enter another. Once a valid name is entered CREATE prints the menu of options (Module 1.2 C_Menu) available and implements the desired option (Module 1.3 Implement) unless the quit option was selected in which case the MDCLL system is exited. The modules of the CREATE program library are shown in Figure 4.2 and described in the following paragraphs.

4.3.1 Name Cell (Module 1.1).

Name_Cell is used throughout the MDCLL system to obtain the name of a cell from the user and to translate this name to the appropriate filename (i.e. add the ".CIF" extension to the cellname). This module also strips off any disk drive letter from the cellname while leaving this prefix on the filename. This module then checks the disk drive to see if the named file is already on disk or not. This on_disk flag, along with the filename and cellname, are then passed back to the calling program.

```
CREATE
   |
   |-- Name_Cell (Para 4.3.1)
   |
   |-- C_Menu (Para 4.3.2)
   |    |-- Get_Option (Para 4.3.2.1)
   |
   |-- Implement (Para 4.3.3)
        |
        |-- Def_Port (Para 4.3.3.1)
        |    |-- Def_Layer (Para 4.3.3.1.1)
        |    |-- Get_Num (Para 4.3.3.1.2)
        |    |-- Port_Tree (Para 4.3.3.1.3)
        |
        |-- Def_Rectangle (Para 4.3.3.2)
        |    |-- Rect_Tree (Para 4.3.3.2.1)
        |
        |-- Def_Wire (Para 4.3.3.3)
        |    |-- S_Wire_Tree (Para 4.3.3.3.1)
        |    |-- Get_W_Option (Para 4.3.3.3.2)
        |    |-- Place_Wire_Parameters (Para 4.3.3.3.3)
        |
        |-- Def_Via (Para 4.3.3.4)
        |    |-- Via_Tree (Para 4.3.3.4.1)
        |
        |-- Def_Flash (Para 4.3.3.5)
        |
        |-- Print_Work (Para 4.3.3.6)
        |    |-- Walk_P_Tree (Para 4.3.3.6.1)
        |    |-- Walk_R_Tree (Para 4.3.3.6.2)
        |    |-- Walk_W_Tree (Para 4.3.3.6.3)
        |    |    |-- Segment_Tree (Para 4.3.3.6.3.1)
        |    |-- Walk_Via_Tree (Para 4.3.3.6.4)
        |    |-- Walk_C_Tree (Para 4.3.3.6.5)
        |    |-- Walk_I_Tree (Para 4.3.3.6.6)
        |
        |-- Save_Cell (Para 4.3.3.7)
             |-- Store_Ptrs (Para 4.3.3.7.1)
```

Figure 4.2 CREATE Program Library Modules

4.3.2 C Menu (Module 1.2).

The C_Menu module displays the available options to the user and then requests the user to pick one of the options (Get_Option Module

1.2.1) based on a one character entry. It accepts the cellname as its only input, simply to display with the menu so as to appear more friendly to the user. If the selected character is not a valid option, an error message stating so is printed along with the menu and the user is asked to choose again. Once a valid option is obtained, it is passed back to the calling module.

4.3.2.1 Get Option (Module 1.2.1).

The Get_Option module is used throughout the MDCLL system to obtain a one character entry from the user. After getting the single character from the user it is converted to upper case. If the entered character was not a return then the module continues getting characters until a return is encountered. This is to ensure no stray characters are left in the machines buffer for the next time an input from the user is needed.

4.3.3 Implement (Module 1.3).

The Implement module simply acts as a clearing house where the option, filename, and cellname are passed into the module, and the appropriate function is then performed. The functions available are to define an interconnection port (Def_Port Module 1.3.1), define a rectangle (Def_Rectangle Module 1.3.2), define a wire (Def_Wire Module 1.3.3), define a via (Def_Via Module 1.3.4), define a round flash (Def_Flash Module 1.3.5), print all the work accomplished on the present cell as stored in the structures (Print_Work Module 1.3.6), and save the cell to disk (Save_Cell Module 1.3.7).

4.3.3.1 Def Port (Module 1.3.1).

The Def_Port module allows the user to define the parameters that make up an interconnection port. These are the portname, the layer the port is on (Def_Layer Module 1.3.1.1), and the x,y location (Get_Num Module 1.3.1.2) of the port. The defined port is then stored in the linked list of inter_port structures (Port_Tree Module 1.3.1.3). The user is then asked if there are more ports to be place and if the answer (Get_Option Module 1.2.1) is no this module is exited. This function is not presently used in the MDCLL system but is envisioned to allow easy interconnection of cells based on port names and eventually automatic routing.

4.3.3.1.1 Def Layer (Module 1.3.1.1).

The Def_Layer module allows the user to define the desired layer. If the current layer is valid for the next function to be performed, based on the input parameter layer_valid, then this module asks if the default layer (the current layer) is wanted. If the default layer is wanted the module exits without making any changes, otherwise it sets the layer to non-valid and, while the layer isn't valid, the optional layers are printed and an option is obtained (Get_Option Module 1.2.1). Finally the global variable "layer" is set to the valid layer selected.

4.3.3.1.2 Get Num (Module 1.3.1.2).

The module Get_Num obtains a string from the user or a file one ASCII character at a time, based on the input parameter "fp". This process continues until either a space, comma, semicolon, return, or a close parenthesis is encountered. This ensures compatibility when getting numbers from a CIF file on disk. This string is then converted from ASCII to integer using the C command "atoi". If the input is coming from the terminal, and the last entered character was not a return, then the module continues getting characters until a return is encountered. This is to ensure no stray characters are left in the machine's buffer for the next time an input from the user is needed. The integer is then passed back to the calling program.

4.3.3.1.3 Port Tree (Module 1.3.1.3).

The Port_Tree module places a portname and parameters into the next available port location. It accepts the portname, x and y location, and a pointer to the next port location as input parameters. It first determines if the pointer is pointing to an existing port structure. If it is, then the program is called recursively with the next port pointer being passed as the current port pointer. After finding a location that isn't defined the appropriate amount of memory is allocated and the portname and x,y location is stored in the inter_ports structure.

4.3.3.2 Def Rectangle (Module 1.3.2).

The Def_Rectangle module first requests the user to define which layer is to be used (Def_Layer Module 1.3.1.1) It then obtains, based on user responses to questions, the rectangle parameters height, length, and x,y location of the bottom left corner of the rectangle (Get_Num Module 1.3.1.2). The rectangle parameters are then stored in the rectangle structure (Rect_Tree Module 1.3.2.1) based on the layer in which the rectangle is placed with an entirely separate structure for each layer defined. The user may then place another rectangle in the present layer based on a yes, no response (Get_Option Module 1.2.1). If no more rectangles are to be defined in the present layer then the user may define rectangles in another layer.

4.3.3.2.1 Rect Tree (Module 1.3.2.1).

The Rect_Tree module accepts a rectangles height, length, x and y location, and a pointer to the next rectangle location. If this location is defined (i.e. has a rectangle already placed) then if the x value to be placed is less than the x value in the present location this module is recalled with the pointer to the left tree location as the input pointer; if the x value to be placed is equal to the present x value then the middle tree pointer is used as the input pointer; and if the x value to be placed is greater than the present x value the right tree pointer is used as the next input pointer location. Once a non-defined location is found the module allocates space and then loads the rectangle information into the rectangle structure.

4.3.3.3 Def Wire (Module 1.3.3).

The Def_Wire module allows the user to place wire statements which can not only change directions and widths but also layers anywhere in the statement. First a starting x,y location has to be defined (Get_Num Module 1.3.1.2). This is all the information needed to start the wire structure (S_Wire_Tree Module 1.3.3.1). Next the user can input as many segments as wanted (Get_W_Option Module 1.3.3.2). The layer (Def_Layer

Module 1.3.1.1) and width can be changed and only after a direction and length for the wire segment is defined is the segment saved to memory (Place_Wire_Parameters Module 1.3.3.3). After finishing one wire the user may enter another wire or exit this module (Get_Option Module 1.2.1).

4.3.3.3.1 S Wire Tree (Module 1.3.3.1).

The module S_Wire_Tree is used to initialize a wire statement in memory. The x,y location along with a pointer to a wire location is input to this module. If the location is already defined with a wire start then the x value of the wire start to be placed is compared with the x value of the already stored wire. The appropriate leg of the trinary tree is then used in the next recursive call to this module. Once an empty location is found, memory space is allocated, the x,y location is stored in the structure, and the pointer to the start of this wire's segments is initiallized.

4.3.3.3.2 Get W Option (Module 1.3.3.2).

The Get_W_Option module accepts the present default width and layer as input to print the available options to the user. The user is then asked to make a selection and, once a valid option is chosen, this option is returned to the calling program.

4.3.3.3.3 Place Wire Parameters (Module 1.3.3.3).

The module Place_Wire_Parameters accepts the width, length, direction, layer, and pointer to a segment as input parameters. If the location being examined already contains a segment, then the module is recursively called with the next segment pointer as input. Once an empty location is found this module allocates memory and then stores the wire segment parameters in the wire_parameters structure.

4.3.3.4 Def Via (Module 1.3.4).

The Def_Via module allows the user to define a via in the cell. It first determines a layer for the vias to be defined (Def_Layer Module 1.3.1.1) and then accepts via x and y locations (Get_Num 1.3.1.2). It then places this information in the proper layer via structure in memory based on the value of the x location (Via_Tree Module 1.3.4.1). The user can continue placing vias in the current default layer or switch to a different layer to define vias before exiting to the calling module (Get_Option Module 1.2.1).

4.3.3.4.1 Via Tree (Module 1.3.4.1).

Via_Tree accepts the via x and y location along with a pointer to a location. If the location already contains a via then the Via_Tree module is called recursively with the new location being based on comparison of the x value to be placed with the present x value. Once an empty location is found, memory space is allocated and the via x and y location is placed in the via structure.

4.3.3.5 Def Flash (Module 1.3.5).

The Def_Flash module is not yet implemented. It should be very similar to the Def_Rectangle module except the inputs from the user would be those necessary for defining a round flash in CIF. In particular, it would have to accept the x and y location and the diameter of the flash and then need a sub-module to place the flash parameters in memory similar to the Rect_Tree module. The structure being filled by the round flash information would also need to be defined in the header.

4.3.3.6 Print Work (Module 1.3.6).

The Print_Work module allows the user to see the work accomplished on the particular cell being worked on. It steps the user through each available structure determining if that structure is needed to be displayed (Get_Option Module 1.2.1). The items which can be displayed are the predefined ports (Walk_P_Tree Module 1.3.6.1), the rectangles by layer in which they are defined (Walk_R_Tree Module 1.3.6.2), the wires (Walk_W_Tree Module 1.3.6.3), the vias one layer at a time (Walk_V_Tree Module 1.3.6.4), the call statements (WalK_C_Tree Module 1.3.6.5), and the iterate statements (Walk_I_Tree Module 1.3.6.6). The last two, calls and iterates, are only included here for completeness even though in CREATE no calls or iterates could be placed. This is because the Print_Work module is used throughout the MDCLL system.

4.3.3.6.1 Walk P Tree (Module 1.3.6.1).

The Walk_P_Tree module prints the interconnection port information to the terminal. This is accomplished recursively by using a pointer to the inter_ports structure and then passing in the next port pointer of the structure on subsequent calls to the module thereby walking through the entire linked list.

4.3.3.6.2 Walk R Tree (Module 1.3.6.2).

The Walk_R_Tree module outputs the rectangle structure information to the terminal one layer at a time. To do this, the trinary rectangle structure trees must be recursively walked to ensure no rectangle is missed. First the left sub-tree is traversed, followed by the middle, and finally the right sub-tree.

4.3.3.6.3 Walk W Tree (Module 1.3.6.3).

Walk_W_Tree prints only the initial starting x and y location of the wire to the terminal and then calls another module (Segment_Tree Module 1.3.6.3.1) to display the wire segments. Once again, as in the rectangle printing, the wire structure trinary tree is recursively walked.

4.3.3.6.3.1 Segment Tree (Module 1.3.6.3.1).

The Segment_Tree module prints each segment of the wire statement, one segment at a time, until it reaches the end of the wire. It performs this recursively by passing in the pointer to the next possible segment location. It only displays the width and layer of the current segment if it is different than the width and layer of the previous segment, which are also input as passed parameters.

4.3.3.6.4 Walk Via Tree (Module 1.3.6.4).

The Walk_Via_Tree outputs the via structure information to the terminal one layer at a time. To do this the trinary via structure tree, as in the rectangle structure tree, must be recursively walked to ensure no via is missed. First the left sub-tree is traversed, followed by the middle, and finally the right sub-tree.

4.3.3.6.5 Walk C Tree (Module 1.3.6.5).

The Walk_C_Tree module recursively walks the call_statement trinary tree. This tree however is based on the cells CIF number. This is accomplished by recursively passing in the pointer to first the left sub-tree, the middle sub-tree, and then the right sub-tree. In printing the information some translation has to be performed to display the flipping of modules and the rotating of modules as the user entered the information. After printing each call statement, the module then prints all portname declarations associated with that call.

4.3.3.6.6 Walk I Tree (Module 1.3.6.6).

Ihe Walk_I_Iree module is almost identical to the Walk_C_Tree module. The only difference is, since some of the information needed for the iterate statement is identical to the call, the iterate makes use of the call_statement structure. Because of this an extra pointer reference is used for some of the information.

4.3.3.7 Save Cell (Module 1.3.7).

The Save_Cell module is used throughout the MOCLL system to save a cell from internal memory to disk. To do this it accepts the filename and cellname as passed parameters. It first constructs an argument consisting of the filename, the cellname, and all the structure pointers for user defined information (Store_Ptrs Module 1.3.7.1). This argument is then passed to the SAVECELL program through execution of the SAVECELL program by the Save_Cell module.

4.3.3.7.1 Store Ptrs (Module 1.3.7.1).

The Store_Ptrs module stores the pointers needed to save a cell already defined, in one single argument. It performs this task by accepting the pointer to be stored, along with the argument to be passed to SAVECELL, and the last position entered in the argument. It converts the pointer to ASCII and then attaches the ASCII version to the argument. The argument and last position entered is then passed back the Save_Cell module.

4.4 DPM Main (Module 2.0).

The DPM program is used to delete a cell from a parent cell in the hierarchy, place a cell into a parent cell, or move the position of a cell in the parent cell. The modules of the DPM program library are shown in Figure 4.3 and described in the following paragraphs. The main module performs four basic functions. It first obtains the name of the parent cell from the user (Name_Cell Module 1.1); if the cellname requested is on disk then this parent cell is brought into main memory (Get_Cell Module 2.1), otherwise the user is asked if a new parent cell by the entered name is wanted (Get_Option Module 1.2.1); if the named cell is the correct parent cellname then the menu of options available is displayed (DPM_Menu Module 2.2), and, as long as the user doesn't request to quit the system, the option requested is implemented (D_Implement Module 2.3).

```
DPM Main
   |
   |-- Get_Cell (Para 4.4.1)
   |     |-- In_File (Para 4.4.1.1)
   |     |-- Get_Port (Para 4.4.1.2)
   |     |-- Get_Rect (Para 4.4.1.3)
   |     |-- Get_Wire (Para 4.4.1.4)
   |     |-- Get_Via (Para 4.4.1.5)
   |     |-- Get_Call (Para 4.4.1.6)
   |     |     |-- Get_C_Parameters (Para 4.4.1.6.1)
   |     |-- Get_Iter (Para 4.4.1.7)
   |
   |-- DPM_Menu (Para 4.4.2)
   |
   |-- D_Implement (Para 4.4.3)
         |
         |-- Place_Call (Para 4.4.3.1)
         |     |-- Call_Parameters (Para 4.4.3.1.1)
         |     |-- Call_Tree (Para 4.4.3.1.2)
         |     |-- C_Port_Names (Para 4.4.3.1.3)
         |           |-- Port_Call_Place (Para 4.4.3.1.3.1)
         |
         |-- Mod_Call (Para 4.4.3.2)
         |
         |-- Iterate_Call (Para 4.4.3.3)
         |     |-- Iter_Parameters (Para 4.4.3.3.1)
         |     |-- Iter_Tree (Para 4.4.3.3.2)
         |
         |-- Mod_Iter (Para 4.4.3.3)
```

Figure 4.3 DPM Program Library Modules

4.4.1 Get Cell (Module 2.1).

The Get_Cell module reads a cell, which the calling program has already confirmed is on the disk, from disk into main memory. It performs this by reading the file one line at a time (In_File Module 2.1.1) and examining the contents of the line. It is looking for CIF comment lines describing the starts of the different types of statements available and also the CIF end of cell marker "E". The statements able to be read are MIF interconnection ports (Get_Port Module 2.1.2), rectangles or boxes (Get_Rect Module 2.1.3), MIF wire statements (Get_Wire Module 2.1.4), MIF via statements (Get_Via Module 2.1.5), call statements (Get_Call Module 2.1.6), and MIF iterate statements (Get_Iter Module 2.1.7). The variable "string" is passed to each of these modules so the maximum size of the string being input need only be defined in one place. Also passed to the modules is the pointer to the file to be read.

4.4.1.1 In File (Module 2.1.1).

The In_File module gets characters from the file specified by the input parameter "fp" until it matches the input parameter designating the character to be found (find_char) or until the end of file is reached. The characters are placed in a string which, upon completion, is passed back to the calling module.

4.4.1.2 Get Port (Module 2.1.2).

The module Get_Port is used to read the interconnection port definitions from the file designated by the input parameter "fp" until it reaches the CIF comment line signifying the end of the port definitions. After determining the line being read is a port definition by reading the CIF "94" statement, the portname is read (In_File Module 2.1.1). The x and y location is read (Get_Num Module 1.3.1.2) and finally the port-layer is read. This information is then stored in the inter_ports structure in main memory (Port_Tree Module 1.3.1.3).

4.4.1.3 Get Rect (Module 2.1.3).

The Get_Rect module is used to read the rectangle (or box) definitions from the file designated by the input parameter "fp". This module starts by getting the first character of the next line of the file. If the character signifies the start of a comment then the comment line is read (In_File Module 2.1.1) and if the comment signifies the end of the rectangle definitions the module is exited. If the character signifies a new layer definition then the layer is read. Lastly, if the character signifies a rectangle (or box) definition, then the rectangle height, length, and x and y location is obtained (Get_Num Module 1.3.1.2). Finally the rectangle is placed in the proper layer rectangle structure (Rect_Tree Module 1.3.2.1).

4.4.1.4 Get Wire (Module 2.1.4).

The Get_Wire module reads MIF wire statements from the CIF file specified by the input parameter "fp". It starts by reading the first word of a CIF comment line (In_File Module 2.1.1). If this first word signifies the end of the wire statements then the module is exited. If the word specifies a wire statement then the x and y location is read (Get_Num Module 1.3.1.2); this information is placed in the wire structure (S_Wire_Tree Module 1.3.3.1); and while there are more segments defined, the segment layer, width, direction, and length are input and then placed in the wire_parameters structure associated with the wire start (Place_Wire_Parameters Module 1.3.3.3).

4.4.1.5 Get Via (Module 2.1.5).

The Get_Via module is used to input MIF via statements by reading CIF box (rectangle) definitions from the file designated by the input parameter "fp". It starts by reading the first character of the line. If this character is a comment marker, then the entire line is input. If the line signifies the end of the via definitions, then this module is exited. If the comment line is the start of the vias in a new layer, then while there are more vias in this layer, the vias x and y locations are read in and the via is stored in the proper structure of via definitions based on layer.

4.4.1.6 Get Call (Module 2.1.6).

The Get_Call module inputs call statements from the file designated by the input parameter "fp". The module starts by reading the first word of the next line (In_File Module 2.1.1). If the word signifies the end of the call statements, then this module is exited. If the first word is the start of a MIF call comment, then the cellname, x and y location, and the bounds of the called cell are input (Get_Num Module 1.3.1.2). If the first word is the start of a CIF call statement, then the call parameters are obtained (Get_C_Parameters Module 2.1.6.1) and the parameters from the MIF call comment and the CIF call statement are placed in the call_statements structure (Call_Tree Module 2.3.1.2). If the first word is a port definition, then the port name is read and placed in the named_ports structure (Port_Call_Place Module 2.3.2.3.1).

4.4.1.6.1 Get C Parameters (Module 2.1.6.1).

The Get_C_Parameters module gets call parameters from the file designated by the input parameter "fp". It first obtains the CIF number and the x and y translational position of the cell being called (Get_Num Module 1.3.1.2). Then, the mirroring and rotational information is decoded from the CIF call statement into the MIF form for storage. This information is then passed back to the calling module

4.4.1.7 Get Iter (Module 2.1.7).

The Get_Iter module is very similar to the Get_Call module. It inputs iterate statements from the file designated by the input parameter "fp". The module starts by reading the first word of the next line (In_File Module 2.1.1). If the word signifies the end of the iterate statements, then this module is exited. If the first word is the start of a MIF iterate comment, then the cellname, number of iterations in the x and y direction, the x and y pitch or spacing, the x and y location, and the bounds of the iterated cell are input (Get_Num Module 1.3.1.2). It then reads characters until the start of the first call statement forming the iteration. From this call statement the parameters which make up a call are obtained (Get_C_Parameters Module 2.1.6.1). Then, the parameters from the MIF iterate comment and the CIF call statement are placed in the iter_statements structure (Iter_Tree Module 2.3.3.2) and the associated call statements structure (Call_Tree Module 2.3.1.2). If the first word is a port definition, then the port name is read and placed in the named_ports structure (Port_Call_Place Module 2.3.2.3.1).

4.4.2 DPM Menu (Module 2.2).

The DPM_Menu module displays the available options to the user and then requests the user to select one of the options (Get_Option Module 1.2.1) based on a one character entry. It accepts the cellname of the parent cell as its only input simply to display with the menu so as to appear more friendly to the user. If the selected character is not a valid option, an error message stating so is printed along with the menu. The user is then asked to choose again. Once a valid option is obtained, it is passed back to the calling module.

4.4.3 D Implement (Module 2.3).

The D_Implement module accepts the option to be performed (as obtained from the DPM_Menu module and the parents cellname as input parameters. If the selected option is to place or modify a call or iterate statement, the name of the cell is first obtained (Name_Cell Module 1.1) and the CIF number is determined (Det_Cifnum Module 7.1). If the option is to place a call or iterate statement and the file named is on disk, then the bounds are obtained from the disk file. If the file is not on disk, the user is asked if the cellname specified is correct (Get_Option 1.2.1). If the cell is on disk, or is to be placed even if not on disk, then either place the call (Place_Call Module 2.3.1) or place the iterate statement (Iterate_Call Module 2.3.3) is executed based on the option requested. If the option was to modify or delete a call or iterate statement, then the proper module is called (Mod_Call Module 2.3.2 Mod_Iter Module 2.3.4). If the option was to print the work, then the work accomplished and stored in main memory for this cell is printed (Print_Work Module 1.3.6). Finally, if the option was to save the cell, then the cell is saved (Save_Cell Module 1.3.7).

4.4.3.1 Place Call (Module 2.3.1).

The Place_Call module accepts the bounds, CIF number, and cellname from the calling module. It then gets the parameters for the call statement from the user (Call_Parameters Module 2.3.1.1) and places them in the call_statements structure in main memory (Call_Tree Module 2.3.1.2). It next gets the port names associated with this call statement (C_Port_Names Module 2.3.1.3). The user is then asked if the present cell should be placed in another location and if not this module is exited (Get_Option Module 1.2.1).

4.4.3.1.1 Call Parameters (Module 2.3.1.1).

The Call_Parameters module obtains the parameters needed for the call statement from the user. It accepts the cellname from the calling module to display with the questions so as to appear more friendly for the user. It obtains the x and y location of the bottom left corner of where the cell is to be placed (Get_Num Module 1.3.1.2), and then determines if the cell is to be flipped upside down and/or left to right (in CIF language mirroring instructions). Finally the user may rotate the x axis of the cell being called to the 3, 6, 9, or 12 o'clock position. This is consistent with the CLL language for laying out cells. Upon saving call statements (see Out_Call_Tree Module 7.3.5), this information is ordered in the CIF call statement with the flipping (mirroring) occurring first, the rotation taking place second, and then finally the translation. The translational position is adjusted so that after the flipping, and the rotation, the new final bottom left corner of the cell is placed where the user defined the bottom left corner to be.

4.4.3.1.2 Call Tree (Module 2.3.1.2).

The Call_Tree module places the information for the call statement in the next available call_statement location. It accepts the cellname, CIF number, translation, mirroring, rotation, bounds, and a pointer to the next call_statement location as input parameters. If the location defined by the pointer already has a call statement residing in it, then this module is recursively called. If the CIF number of the cell to be placed is greater than the CIF number of the present call, then the left tree pointer is used as the subsequent input. If the CIF number of the cell to be placed is equal to the present CIF number, then the middle tree pointer is used. Finally, if the CIF number is greater, then the right tree pointer is used. After a location is found which is not defined, this module allocates space and the information is loaded into the call_statement structure.

4.4.3.1.3 C Port Names (Module 2.3.1.3).

The module C_Port_Names allows the user to define as many portnames as needed to be associated with the call statements. The user simply enters the port name which is then placed in the named_port structure with the call statement (Port_Call_Place Module 2.3.1.3.1). The user is then asked if any more ports are to be defined and if not the module is exited (Get_Option 1.2.1).

4.4.3.1.3.1 Port Call Place (Module 2.3.1.3.1).

The Port_Call_Place module accepts the portname and pointer to the next port location as inputs. If the location is already filled, then the module is recursively called using the next port location as input until an empty location is found. Once an empty location is found, memory space is allocated and the portname is stored in the named_ports structure.

4.4.3.2 Mod Call (Module 2.3.2).

The Mod_Call module allows the user to modify or delete call statements already defined. It accepts as input an exit control flag, a CIF number, and a pointer to the next call statement. If the location defined by the pointer is not empty and the exit flag is false, then the module checks if the input CIF number is less than or equal to the present CIF number (this is done so the user need not enter the cellname to be modified, which is then translated to the CIF number, to modify the call statements). If the CIF number is valid, then the call statement is displayed for the user. The user may then either modify this call statement, delete this call statement, go to the next call statement, or exit this module (Get_Option Module 1.2.1). If the option is to exit the module then the exit flag is set to true and the module is exited. If the option is to delete the call statement, then the call is deleted by setting the y translation parameter to "-9999". (The structure definition needs to be maintained with the CIF number to keep the trinary tree structure intact without massive reorganization of the other called cells.) If the option is to modify the call statement, then all the call parameters must be re-entered (Call_Parameters Module 2.3.1.1). After each call statement the user can modify or delete the port names associated with the call (C_Port_Names Module 2.3.1.3). The options continue until all the call statements have been processed or the exit command is given.

4.4.3.3 Iterate Call (Module 2.3.3).

The Iterate_Call module is very simplistic. It accepts the bounds, CIF number, and cellname as input and simply calls five other modules. First it gets the call parameters associated with this iterate statement (Call_Parameters Module 2.3.1.1). It then obtains the iteration parameters (Iter_Parameters Module 2.3.3.1), places the iteration parameters in the iter_statement structure (Iter_Tree Module 2.3.3.2), and the associated call statement parameters in the associated call_statement structure (Call_Tree Module 2.3.1.2). Finally, it gets the port names to be associated with this iteration (C_Port_Names Module 2.3.1.3).

4.4.3.3.1 Iter Parameters (Module 2.3.3.1).

The module Iter_Parameters allows the user to input the parameters for iterating a cell. The bounds of the cell are passed into this module. First the user is requested to enter the number of iterations in the x and y direction (Get_Num Module 1.3.1.2). Then the user may define an x and y pitch, or spacing, for the iterated cells or default to the size of the bounding box as determined by the bounds input to this module. The iteration and the pitch information are then passed back to the calling module.

4.4.3.3.2 Iter Tree (Module 2.3.3.2).

The Iter_Tree module places the information for the iterate statement in the next available iter_statement location. It accepts the cellname, CIF number, number of iterations in the x and y direction, the x and y pitch, and a pointer to the next iter_statement structure as input parameters. If the location defined by the pointer already has an iteration residing in it, then this module is recursively called. If the CIF number of the cell to be iterated is greater than the CIF number of the present iterated cell, then the left tree pointer is used as the subsequent input. If the CIF number of the cell to be iterated is equal to the present CIF number, then the middle tree pointer is used. If the CIF number is greater, then the right tree pointer is used. Once a non-defined location is found, the module allocates space and the information is loaded into the iter_statement structure.

4.4.3.4 Mod Iter (Module 2.3.4).

The Mod_Iter module allows the user to modify or delete iterate statements previously defined. It accepts as input an exit control flag, a CIF number, and a pointer to the next iterate statement. If the location defined by the pointer is not empty and the exit flag is false, then the module checks if the`input CIF number is less than or equal to the present CIF number (this is done so the user need not enter the cellname of the iteration to be modified] which is translated to the CIF number, to modify iterate statements). If the CIF number to be found is less than or equal to the CIF number of the iteration statement being examined, then the iterate statement is displayed for the user. The user may then either modify or delete this iterate statement, go to the next iterate statement, or exit this module (Get_Option Module 1.2.1). If the option is to exit the module, then the exit flag is set to true and the module is exited. If the option is to delete the iterate statement, then the iteration is deleted by setting the number of x iterations to "-9999". (The structure needs to be maintained with the CIF number to keep the trinary tree structure intact without massive reorganization of the other iterations.) If the option is to modify the iterate statement, then all the iterate parameters and the associated call parameters must be re-entered (Iter_Parameters Module 2.3.3.1 and Call_Parameters Module 2.3.1.1). After each iterate statement the user can modify or delete the port names associated with that iteration (C_Port_Names Module 2.3.1.3). The options continue until all of the iterate statements have been processed or the exit command is given.

4.5 MODIFY Main (Module 3.0).

The MODIFY program is used to modify a previously defined cell. The main program first obtains the name of the cell to be modified from the user (Name_Cell Module 1.1). If the cell can't be found on disk, the user is asked if he wants to try again (Get_Option Module 1.2.1). If not, the program is exited. Once a valid file is obtained the cell is read into main memory from disk (Get_Cell Module 2.1). Next, it is determined if the user wants to add to this cell, modify (or delete) cell parameters, or quit to the system (Det_Mod Module 3.1). If the user wants to add to the module then the menu used for creating a cell is displayed (C_Menu Module 1.2) and while the selected option is other than to quit, the option is implemented (Implement Module 1.3). On the other hand, if the user wants to modify the cell a menu is displayed showing the options available, and, while the option is other than to quit, the option is implemented (M_Implement Module 3.3). The modules of the MODIFY program library are shown in Figure 4.4 and are described in the following paragraphs.

```
MODIFY Main
  |
  |-- Det_Mod (Para 4.5.1)
  |
  |-- M_Menu (Para 4.5.2)
  |
  |-- M_Implement (Para 4.5.3)
       |
       |-- Mod_Port (Para 4.5.3.1)
       |-- Mod_Rect (Para 4.5.3.2)
       |-- Mod_Wire (Para 4.5.3.3)
       |-- Mod_Via (Para 4.5.3.4)
       |-- Mod_Flash (Para 4.5.3.5)
```

Figure 4.4 MODIFY Program Library Modules

4.5.1 Det_Mod (Module 3.1).

The Det_Mod module allows the user the selection of adding to a cell, modifying a cell, or quitting to the operating system based on a one character entry (Get_Option Module 1.2.1). If the selected character is not a valid option then the user is so told and requested to select again. Once a valid option is selected the option is passed back to the calling module.

4.5.2 M Menu (Module 3.2).

The M_Menu module displays the available options to the user and then requests the user to pick one of the options (Get_Option Module 1.2.1) based on a one character entry. It accepts the cellname as its only input simply to display with the menu so as to appear more friendly to the user. If the selected character is not a valid option an error message stating so is printed along with the menu and the user is asked to choose again. Once a valid option is obtained, it is passed back to the calling module.

4.5.3 M Implement (Module 3.3).

The module M_Implement accepts the option to be performed, the filename, and the cellname as input parameters. If the option is to print, then the cell, as currently stored in memory as opposed to the version on disk, is displayed to the user (Print_Work Module 1.3.6). If the option is to save the cell to disk, then the cell as currently stored in memory is saved (Save_Cell Module 1.3.7). If the user wants to modify a statement, other than a call, iteration, or wire, the layer needs to be determined (Def_Layer Module 1.3.1.1) and an optional x location (Get_Num Module 1.3.1.2), which is used to go directly to the statement to be modified. If the user wants to modify port definitions for this cell, then the modify port module is called (Mod_Port Module 3.3.1). If the user wants to make modifications to the rectangles defined, then call the modify rectangle module with the proper layer structure in which they are defined (Mod_Rect Module 3.3.2). If the option is to modify a wire, then enter the module to modify a wire

(Mod_Wire 3.3.3). If the option is to modify a via, then enter the modify via module with the tree structure of the layer of vias requested (Def_Via Module 3.3.4). Finally if a flash needs modification, then enter the module to modify round flashes (Mod_Flash Module 3.3.5).

4.5.3.1 Mod Port (Module 3.3.1).

The Mod_Port module allows the user to modify or delete previously defined interconnection ports. This module accepts a pointer to the next inter_ports structure as input, and if there is a port at that location, the port parameters are displayed to the user. The user may then either exit this module, delete the port definition displayed, modify the port, or go to the next port in the structure. If the option is to delete the port then the y location of the port is set to "-9999" (since the location in the structure must be maintained to avoid reorganization of the subsequent port definitions). If the user wants to modify the port, then the new portname, layer (Def_Layer Module 1.3.1.1), and x and y location (Get_Num Module 1.3.1.2) are defined. If the option selected was other than to exit this module, the module is recursively called with the next port location until the inter_ports structure is exhausted.

4.5.3.2 Mod Rect (Module 3.3.2).

The Mod_Rect module allows the user to modify or delete rectangle definitions previously entered. It accepts as input an exit control flag, an x location, and a pointer to the next rectangle in the structure. If the location defined by the pointer is not empty and the exit flag is false, then the module checks if the input x location is less than or equal to the present rectangle's x location (this is performed so the user may step through all the rectangles if desired, or simply go directly to the rectangle to be modified). If the x location to be found is less than or equal to the x location of the rectangle statement being examined, then the rectangle statement is displayed for the user. The user may then either modify this rectangle, delete this rectangle, go to the next rectangle in the structure, or exit this module (Get_Option Module 1.2.1). If the option is to exit the module, then the exit flag is set to true and the module is exited. If the option is to delete the rectangle, then the rectangle is deleted by setting the height and length of the rectangle to zero. (The structure needs to be maintained with the x location to keep the trinary tree structure intact without massive reorganization of the other rectangles in this layer.) If the option is to modify the rectangle, then the height, length, and x and y location of the rectangle must be re-entered (Get_Num Module 1.3.1.2). If the x location has changed, then the rectangles present location is deleted as described above and a new rectangle is placed in the structure (Rect_Tree Module 1.3.2.1). The options continue until all the rectangles in the structure have been processed or the exit command is given.

4.5.3.3 Mod Wire (Module 3.3.3).

The Mod_Wire module allows the user to modify or delete wire statements previously entered. It accepts as input an exit control flag, an x location, and a pointer to the next wire statement in the structure.

Once a valid wire is found, by use of the input x location as described in the previous paragraph, the wire statement is displayed. The user may then either modify this wire, delete this wire, go to the next wire in the structure, or exit this module (Get_Option Module 1.2.1). If the option is to exit the module, then the exit flag is set to true and the module is exited. If the option is to delete the entire wire, then the wire is deleted by erasing the segments associated with this wire start. (The structure needs to be maintained with the x location to keep the trinary tree structure intact without massive reorganization of the other wires.) If the option is to modify the wire, then the new x and y location of the wire start is obtained (Get_Num Module 1.3.1.2). If the x location has changed, then the wire's present location is deleted as described above and a new wire is placed in the structure (S_Wire_Tree Module 1.3.3.1) along with all of its previously defined segments (Place_Wire_Parameters Module 1.3.3.3). If the x location hasn't changed, then one at a time the user may modify the individual segments' width, length, and/or layer. The options continue until all the wires in the structure have been processed or the exit command is given.

4.5.3.4 Mod Via (Module 3.3.4).

The module Mod_Via allows the user to modify or delete via statements previously entered. It accepts as input an exit control flag, an x location, and a pointer to the next via in the structure. It displays the valid vias to the user based on the input x location in the same manner as the Mod_Rect module. The user may then either modify this via, delete this via, go to the next via in the structure, or exit this module (Get_Option Module 1.2.1). If the option is to exit the module, then the exit flag is set to true and the module is exited. If the option is to delete the via, then the via is deleted by setting the y location of the via to "-9999". If the option is to modify the via then the x and y location of the via must be re-entered (Get_Num Module 1.3.1.2). If the x location has changed, then the via's present location is deleted as described above and a new via is placed in the structure (Via_Tree Module 1.3.4.1). The options continue until all the vias in the structure have been processed or the exit command is given.

4.5.3.5 Mod Flash (Module 3.3.5).

The Mod_Flash module is not yet implemented. It should be very similar to the Mod_Rect module except the modifications would be to those parameters defining a round flash. In particular the x and y location and the diameter of the flash would have to be allowed to be modified or the entire flash deleted.

4.6 INTER Main (Module 4.0).

The INTER program is intended to be used to interconnect the cells in a circuit. It has not yet been implemented. It is envisioned to allow the user to route not only single wires, but also bus lines, around a circuit more easily than placing the wires one at a time with the wire statement as is currently suggested in the MDCLL system. It is also hoped that an automated router can be configured into this system. In either case, the use of the interconnection ports should provide an easier method for connecting the cells making up a circuit.

4.7 PRIPLO Main (Module 5.0).

The PRIPLO program, presently, only performs one function. This is to print a cell's file as contained on disk to the user's terminal. It is envisioned, however, that this program should be expanded to also allow the file to be output to a printer. Also, the program should provide for the quick plotting of a simple bounding box type plot of a cell to a standard printer (so any printer could handle the output, using only the standard character set). Admittedly this would cause some of the detail to be lost, but quite often the user simply needs an overall picture of the cell in question.

4.8 FINALIZE Main (Module 6.0).

The FINALIZE program is used to combine all the cells referenced (called or iterated), by the highest level cell in a design, into one file for subsequent plotting using MCIFPLOT or to be sent off for production. The modules in the FINALIZE program library are shown in Figure 4.4 and described in the paragraphs below. First, the user must enter the name of the cell to be finalized (Name_Cell Module 1.1). If the file is not found on disk the user is given the option of trying again (Get_Option Module 1.2.1). If the file is on disk then the cell's calls to other cells and iterations of other cells must be obtained and stored for later processing (In_File Module 2.1.1, Get_Call Module 2.1.6, and Get_Iter Module 2.1.7). Once all the references to other cells are obtained from the cell to be finalized, the final CIF is then formatted (Format_Cif Module 6.1). Finally control is returned to the MDCLL program (MDCLL Main Module 0.0).

```
FINALIZE Main
  |
  |-- Format_Cif (Para 4.8.1)
          |
          |-- Primary_Calls (Para 4.8.1.1)
          |     |-- Call_Names (Para 4.8.1.1.1)
          |
          |-- Iter_Calls (Para 4.8.1.2)
          |
          |-- Process_Call (Para 4.8.1.3)
```

Figure 4.5 FINALIZE Program Library Modules

4.8.1 Format Cif (Module 6.1).

The module Format_Cif accepts the filename and cellname of the cell to be finalized. It then places all the cellnames of the primary calls and the primary iterated calls into a single linked list. (Primary_Calls Module 6.1.1 and Iter_Calls Module 6.1.2). The linked list of cellnames is then processed by reading the file from disk, placing any called or iterated cells names into the linked list, and writing the file to the "final.cif" file on disk. While there are more cellnames in the called cells linked list, the referenced cell is processed in this manner (Process_Call Module 6.1.3). Finally, the parent cell being finalized is written to the "final.cif" file (In_File Module 2.1.1 and Out_File Module 7.2.1).

4.8.1.1 Primary Calls (Module 6.1.1).

The Primary_Calls module is very simplistic. It accepts a pointer to the next call statement as input and recursively steps through the call_statement structure placing the called cells names into the called_ cells linked list structure (Call_Names Module 6.1.1.1). This module need only process the called cells with CIF numbers either greater than or less than the previous cell since no duplication of cells is needed.

4.8.1.1.1 Call Names (Module 6.1.1.1).

The Call_Names module accepts a cellname, CIF number, and pointer to the next called name as input. The module then searches for either an already existing storage of the cell being referenced, or if there is no previous reference, then memory is allocated and the cellname and CIF number are stored in the called_cells structure. In this manner, each called cell is only defined once in the final CIF file regardless of the number of times it is invoked.

4.8.1.2 Iter Calls (Module 6.1.2).

The Iter_Calls module is very simplistic and is identical to the Primary_Calls module, except that it walks the iter_statement structure. It accepts the pointer to the next iterate statement as input and recursively steps through the iter_statement structure placing the iterated cells names into the called_cells linked list structure (Call_Names Module 6.1.1.1). This module need only process the iterated cells with CIF numbers either greater than or less than the previously processed cell since no duplication of cells is needed.

4.8.1.3 Process Call (Module 6.1.3).

The Process_Call module accepts the cellname to be processed and the file pointer to the "final.cif" file to which the cell should be written as input via passed parameters. It then converts the cellname to the needed filename and opens the cell's file to be processed. If the file can't be opened, the user is asked if the module should try to open the file again, thereby allowing a change of disks (Get_Option Module 1.2.1). Once the file is opened, the file is copied, one line at a time, to the "final.cif" file (In_file Module 2.1.1 and Out_File Module 7.2.1). If the line processed is either a reference to a call statement or an iteration, then the associated cellname and CIF number are added to the called_names structure (Call_Names Module 6.1.1.1 and Det_Cifnum Module 7.1).

4.9 SAVECELL Main (Module 7.0).

The SAVECELL program is used to save a cell from main memory to disk. It accepts as input, a single string argument which contains the filename, cellname, and pointers to the structures to be converted to CIF and saved to disk. The program performs four basic functions: it determines the CIF number of the cell to be saved (Det_Cifnum Module 7.1); it then writes the CIF header to the file on disk (Header_Out_File Module 7.2); the CIF cell layout is then written to the file (Out_Cell_Layout Module 7.3); finally, the CIF footer is written to the file on disk. After completing the cell, the MDCLL program is executed (MDCLL Main Module 0.0). The modules in the SAVECELL program library are shown in Figure 4.5 and are described in the following paragraphs.

```
SAVECELL Main
  |
  |-- Det_Cifnum (Para 4.9.1)
  |
  |-- Header_Out_File (Para 4.9.2)
  |     |-- Out_File (Para 4.9.2.1)
  |
  |-- Out_Cell_Layout (Para 4.9.3)
  |     |
  |     |-- Out_Port_Tree (Para 4.9.3.1)
  |     |-- Out_Rect_Tree (Para 4.9.3.2)
  |     |-- Out_Wire_Tree (Para 4.9.3.3)
  |     |     |-- Out_W_Comment (Para 4.9.3.3.1)
  |     |     |-- Out_W_Segment (Para 4.9.3.3.2)
  |     |-- Out_Via_Tree (Para 4.9.3.4)
  |     |-- Out_Call_Tree (Para 4.9.3.5)
  |     |     |-- Out_C_Statement (Para 4.9.3.5.1)
  |     |     |-- Out_P_Names (Para 4.9.3.5.2)
  |     |-- Out_Iter_Tree (Para 4.9.3.6)
  |
  |-- Footer_Out_File (Para 4.9.4)
```

Figure 4.6 SAVECELL Program Library Modules

4.9.1 Det Cifnum (Module 7.1).

The module Det_Cifnum determines the CIF number for the cell. It performs this task by accepting the name of the cell as an input parameter and then, one letter at a time, creating an integer based on the value and the position of the character. It then insures that this integer is non-negative and is greater than one thousand. The final calculated CIF number is then returned to the calling program.

4.9.2 Header Out File (Module 7.2).

The Header_Out_File sets up the starting information needed to define a CIF cell. This includes a "DS" (or Definition Start) statement, followed by the CIF number, and scaling factor of CIF to lambda units. Then a comment is included giving the title of the cell.

4.9.2.1 Out File (Module 7.2.1).

Out_File accepts two strings and a pointer to the file to be written to as input parameters. It first outputs the one string, and then the other to the file specified. This module was implemented in this manner since it seemed every time this module was invoked a variable and a separator, of more than one character, had to be output. This method reduced, by two, the number of calls to this module.

4.9.3 Out Cell Layout (Module 7.3).

Out_Cell_Layout controls the output of the cell to the file designated by the input parameter "fp". It first outputs any ports that are defined in the inter_ports structure (Out_Port_Tree Module 7.3.1). It then outputs the rectangles defined, one layer at a time (Out_Rect_Tree Module 7.3.2). It then outputs the MIF wire comments (Out_Wire_Tree Module 7.3.3). This is followed by the rectangles which make up the wire statements, once again output one layer at a time (Out_Rect_Tree Module 7.3.2). Then the vias making up the wire statements are output (Out_Via_Tree Module 7.3.4). Since there are three layers associated with each via defined in a given layer, the vias are output by the layer in which they are defined and each of the three layers associated with the vias. This causes a few extra CIF layer definition statements, but the output routine is consistent with the construction of the via. Next, the vias defined by the user are output to the file (Out_Via_Tree Module 7.3.4) followed by the call statements contained in the call_statements structure (Out_Call_Tree Module 7.3.5), and finally the iterate statements (Out_Iter_Tree Module 7.3.6).

4.9.3.1 Out Port Tree (Module 7.3.1).

The Out_Port_Tree module recursively steps through the inter_ports structure outputting all the ports which were not deleted. It accepts the pointer to the next port location and the pointer to the output file as passed parameters. To output a port, it first outputs a "94", signifying a statement to be printed, followed by the x and y location and the layer of the port.

4.9.3.2 Out Rect Tree (Module 7.3.2).

The module Out_Rect_Tree accepts the pointer to the next rectangle structure and the pointer to the output file as input parameters. If the rectangle hasn't been deleted, then the CIF box statement is output to the file. This includes the x and y location of the middle of the rectangle, the height of the rectangle, and the length of the rectangle. Then, if needed, the bounds of the cell are adjusted. The trinary rectangle tree structure is recursively walked until all rectangles have been processed.

4.9.3.3 Out Wire Tree (Module 7.3.3).

The Out_Wire_Tree module accepts the pointer to the next wire statement and the pointer to the output file as input parameters. It outputs a CIF comment line containing the start of the wire followed by the wire segments (Out_W_Comment Module 7.3.3.1), and then processes the MDCLL wire statement into MDCLL rectangles and vias (Out_W_Segment Module 7.3.3.1). Both modules are performed recursively until every wire in the trinary wire_statement structure tree has been processed.

4.9.3.3.1 Out W Comment (Module 7.3.3.1).

The Out_W_Comment module simply outputs the segments of the wire out to the disk in the form of a CIF comment line. This is performed recursively until every segment is output. It accepts the previous segments width and layer as input along with a pointer to the next segment to be processed and the pointer to the output disk file.

4.9.3.3.2 Out W Segment (Module 7.3.3.2).

The Out_W_Segment module is the most complicated of all the output modules since it determines the location and size of the rectangles making up a wire, and the vias allowing any necessary layer changes in the wire. It accepts the previous segment's width and layer, as well as the x and y location for the start of this segment, the pointer to the next wire segment, and the pointer to the output file. It first outputs the comment describing the segment. Next, it determines the proper rectangle parameters to describe this segment and places them in the rectangle structure by layer (Rect_Tree Module 1.3.2.1). Then, if there is a next segment, the x and y location is adjusted to the opposite end of the present segment, and if the next segment layer is different then the current layer, the proper x and y location of the needed via is stored in the proper via structure based on layer (Via_Tree Module 1.3.4.1). Finally, the starting x and y location for the next segment has to be determined based on the direction of movement to ensure all corners are filled in. This module is called recursively until every segment of the wire is processed.

4.9.3.4 Out Via Tree (Module 7.3.4).

The module Out_Via_Tree outputs one layer of the via definition at a time. It accepts the pointer to the next via, the via size, and the pointer to the output file as input parameters. The via x and y location is output along with the size of the via based on the size given by the input. If needed, the bounds of the cell are adjusted accordingly. The via trinary tree structure is recursively walked ensuring all vias are output.

4.9.3.5 Out Call Tree (Module 7.3.5).

The Out_Call_Tree module accepts the pointer to the next call statement and the pointer to the output file as input parameters. It then outputs a CIF comment line describing the call statement and then the CIF call statement itself (Out_C_Statement Module 7.3.5.1). It then outputs the port names associated with the call (Out_P_Names Module 7.3.5.2). This is performed recursively until all the call statements have been processed.

4.9.3.5.1 Out C Statement (Module 7.3.5.1).

Out_C_Statement outputs the call statements to the output file designated by the input parameter "fp". It also accepts as input the CIF number of the cell, translational co-ordinates, mirroring instructions, rotational position of the x-axis, and the bounds of the called cell. This module outputs the call statement in a set order: first the mirroring is performed, then any necessary rotation, followed by the translation. Based on the mirroring and rotational instructions the final translational position may or may not need to be adjusted. The module ensures that after the mirroring and rotating, the new lower left corner of the cell is placed at the translational location specified by the user. Finally, the bounds of the parent cell are adjusted as needed.

4.9.3.5.2 Out P Names (Module 7.3.5.2).

The Out_P_Names module accepts the pointer to the next named_ports location and the pointer to the output file as input parameters. While there are more port names defined, this module outputs the portname, in the form of a CIF comment, to the file.

4.9.3.6 Out Iter Tree (Module 7.3.6).

The Out_Iter_Tree module accepts the pointer to the next iterate statement and the pointer to the output file as input parameters. It then outputs a CIF comment line describing the iterate statement and then the number of CIF call statements needed to describe this iteration (Out_C_Statement Module 7.3.5.1). This module then outputs the port names associated with the iteration (Out_P_Names Module 7.3.5.2). This is performed recursively to process all the iterate statements.

4.9.4 Footer Out File (Module 7.4).

Finally, the CIF end of cell information is output to the file using the Footer_Out_File module. This module accepts the cellname, CIF number, and pointer to the output file as input parameters. It includes a CIF comment line describing the bounds of the cell, the CIF definition finished marker (DF), a CIF call statement invoking the cell, and the CIF end of file marker "E".

5. Analysis

This analysis examines the degree to which the requirements stated in Chapter 2 were met. The requirements to be checked are the overall requirements, the implementation requirements, the requirements of the system and each function in the system, and finally the performance requirements.

5.1 Overall.

The basic requirement of the system is to be easy to use by the chip designer. This requirement has been achieved by the segmentation of the programs based on function, and the use of menus. The use of independent programs and libraries of routines, along with the implementation of a standard header for use throughout the system, and the inclusion of the hierarchy charts as part of the code has made the system easily maintainable as well.

The next requirement examined is the compatibility with CIF and CLL files. The output files produced by the MDCLL system are totally CIF compatible; however, the MDCLL system cannot read standard CIF files nor files produced by CLL. This is caused by the fact that MDCLL reads comment lines to determine if the CIF statements that follow are MDCLL rectangles, wires, or vias. To solve this problem, the Get_Cell module (2.1) would have to be modified to include the ability to read the CIF layer, box, and call statements. Also, extra pointers would need to be included to allow for boxes (rectangles) in layers which form vias (e.g. glass) and the necessary software to manipulate these boxes throughout the system. However, implementing this would negate any savings in internal memory space for the cell and design time afforded by the MDCLL wire, via, and iterate statements which are much easier to use than the CIF box and call statement.

Table 5.1 shows the CIF commands implemented.

CIF Command	Required	Implemented
Polygon with path		
Box with length, width, center, and direction	X	X
Round Flash	X	
Wire with width and path		1
Layer specification	X	X
Start symbol definition	X	X
Finish symbol definition	X	X
Delete symbol definition		
Call symbol	X	X
Comments	X	2
End marker	X	X

1. A wire statement similar to CLL was implemented and is translated into CIF box statements on output.
2. Comments are included in the CIF produced, but the user cannot enter comment lines.

TABLE 5.1 CIF commands implemented

Another concern for the experienced circuit designer is that the MDCLL system presently only accepts integers when numbers are needed. This limitation of using only integers causes the MDCLL lambda unit to be half the size of the common CLL and CIF lambda unit (1 common CIF lambda unit = 2 MDCLL lambda units), which results in a different scaling factor in the MDCLL system. This use of integers is simpler for both the program and the user since there are no fractions to contend with. Another problem with using only integers is in the conversion to CIF box statements which require the center of the box as reference. To perform this conversion the length and width of the box must be integrally divided in half (difficult to perform with odd integers) and therefore this is another reason for doubling the normal number of lambdas on a grid. If the use of non-integer values was essential, the internal machine representation of the values could remain as integers and simply convert as needed for input and output.

5.2 Implementation.

The MDCLL system was successfully designed and implemented on a CP/M operating system-based microcomputer with 64K of main memory. The largest executable file in the system (see Table 5.2 for the sizes of the executable files) is 26K which, leaving 16K for the operating system

Program	Assembly language	Executable File
	File Size (Kbytes)	(bytes/K bytes)
MDCLL	16 K	5163 / 6 K
CREATE	58 K	24481 / 24 K
DPM	98 K	23255 / 24 K
MODIFY	112 K	26580 / 26 K
INTER	10 K	3683 / 4 K
PRIPLO	16 K	4601 / 5 K
FINALIZE	44 K	10627 / 12 K
SAVECELL	70 K	16099 / 16 K
Total disk space needed		117 K

Table 5.2 MDCLL Disk Space Usage

and 2K as spare, leaves an internal storage area of 20K for any one MDCLL cell (see Paragraph 5.4 for sample cells). As shown in Table 5.2, the total disk space needed for the executable MDCLL files is 117K. (The assembly language size of each program is also given to show the amount of disk space needed for the compilation process.) This 117K easily fits onto one diskette and leaves room for cells on the diskette or the system allows the use of other disk drives for cell file storage.

There are two compiler semi-unique features utilized by the MDCLL system. One is the use of the "exec" command which allows one program to execute another without returning to the first. This function is similar to the "system" command described in Kernighan and Ritchie, however, the "system" command returns control to the calling program. This is semi-unique since most C compilers support this type of function, although the actual command line may be different. The second semi-unique compiler function used is the "#ifneed" and "#endif" statement pair used in the module libraries. This pair of statements allows the inclusion of an entire library of modules while only the modules needed by the program are actually included during compilation. Most C compilers have a function similar to this, although once again, the actual command lines may vary.

5.3 System.

The MDCLL system is capable of allowing the user to create new cells in CIF, delete, place, or move cells in relation to other cells, modify cells, interconnect cells, and print CIF files. The programs implementing these functions are shown in Table 5.3 along with the needed libraries of modules in the MDCLL system. (The functions are described in the following paragraphs and the hierarchy/structure charts and the code are included in Appendices A and B.) The system does not allow the input of a cell from a library of cells, although the cell needed could be copied into a separate file and then used by way of the call or iterate MDCLL statements.

File	Size	Included files
MDCLL.H	6 K	TPRINTF.C, EXEC.C
MDCLL.C	10 K	MDCLL.H, STDLIB.C
CREATE.C	8 K	MDCLL.H, CRE8LIB.LIB, STDLIB.C
CRE8LIB.LIB	44 K	none
DPM.C	10 K	MDCLL.H, DPMLIB.LIB, CRE8LIB.LIB, SAVELIB.LIB, STDLIB.C
DPMLIB.LIB	48 K	none
MODIFY.C	8 K	MDCLL.H, MODLIB.LIB, DPMLIB.LIB, CRE8LIB.LIB, STDLIB.C
MODLIB.LIB	22 K	none
INTER.C	2 K	MDCLL.H
PRIPLO.C	4 K	MDCLL.H, DPMLIB.LIB, CRE8LIB.LIB, SAVELIB.LIB, STDLIB.LIB
FINALIZE.C	8 K	MDCLL.H, FINALLIB.LIB, DPMLIB.LIB, MODLIB.LIB, CRE8LIB.LIB, SAVELIB.LIB, STDLIB.C
FINALLIB.LIB	8 K	none
SAVECELL.C	4 K	MDCLL.H, SAVELIB.LIB, CRE8LIB.LIB, STDLIB.C
SAVELIB.LIB	34 K	none
TPRINTF.C	4 K	This is a system file
EXEC.C	6 K	This is a system file
STDLIB.C	10 K	This is a system library file
Total disk space = 236 K		

Table 5.3 Programs and libraries used in the MDCLL system

5.3.1 Create Function.

The MDCLL system is capable of creating a new CIF file based on user inputs by use of the CREATE and SAVECELL programs. The CREATE program allows the user to define rectangles, wires, vias, and interconnection ports. These definitions are then converted to CIF and stored to disk by the SAVECELL program. The CIF polygon, round flash, and CIF wire statement were not implemented in this version of MDCLL as shown in Table 5.1. However, a wire statement and via similar to the CLL statements are supported. The naming of wires, vias, and rectangles is not supported in this version. This can easily be implemented by adding storage locations in the structures defining these statements, allowing the user to input and modify the information, and providing for the output of this information. When the user is finished creating a cell the cell is converted to CIF and saved to disk by the SAVECELL program.

5.3.2 Delete, Place, or Move Function.

The MDCLL system allows the user to delete, place, or move a cell in the hierarchy of cells through the use of the DPM and SAVECELL programs. Included is the ability to iterate cells in the same manner as in the CLL language. To keep the functions segmented into programs, the DPM program not only includes the creation of calls and iterations but also provides the user with the capability to modify or delete calls and iterations. The requirement of warning the user if cells overlap has not been implemented in this version of MDCLL. When the user is finished, the new or modified cell is converted to CIF and stored to disk by the SAVECELL program.

5.3.3 Modify Function.

The MDCLL system allows the user to modify any cell in the hierarchy of cells by use of the MODIFY and SAVECELL programs. The MODIFY program reads the cell to be modified into main memory and, through the use of menus, allows the user to modify any previous entries and to add new information. Upon completion, the new cell is then converted to CIF and saved to disk by the SAVECELL program. The requirement to inform the user if the cell has changed in size has not been implemented, but could easily be accomplished by storing the previous cell size (or bounds) and comparing this size with the new cell size.

5.3.4 Interconnect Function.

The interconnect function has not been implemented in this version of MDCLL. Upon entrance to the interconnect program (INTER) the user is informed that cell interconnection must be performed by use of the modify function and it is suggested that the wire statement be used. Future versions of MDCLL should allow the interconnection of cells via the use of the predefined port names as the locations for wire starts and stops and eventually even an automatic routing routine.

5.3.5 Print/Plot Function.

The MDCLL system has the requirement to be able to print the CIF files of the cells and plot not only the bounding box diagram of any cell in the hierarchy showing the location of cells within the cell being plotted, but also the entire CIF layout by execution of the MCIFPLOT program. The ability to print files to the terminal has been implemented by use of the PRIPLO program but neither the plotting routine, nor the direct interface with MCIFPLOT, have been implemented in this version of MDCLL. To obtain a plot, the user must learn to use the MCIFPLOT program.

5.4 Performance.

The main performance criteria of the MDCLL system is to be an easy to use system. This translates to menus that are self-explanatory, so the user doesn't get frustrated, and response times that keep the user from becoming bored waiting for the system. The MDCLL system provides both of these. The menus are easily understood and provide simple error messages prompting the user to input correct data. The response time of the system is very good. The system responds immediately to the user on all commands except the following. Each time a different program in the system is executed a noticeable delay is caused by the reading of the program into memory. This delay cannot be alleviated. Another delay is noticed when a cell is input from disk into main memory or a cell is output to disk from main memory (the larger the cell, the longer the delay). To aid in keeping the user from becoming bored at this stage in the system, messages are printed to the screen upon the start of inputting or outputting the next set of CIF statements (e.g. rectangles and vias). The last area where the user may become concerned is during the FINALIZE program which brings several CIF files together into one file. Here, the files are printed to the screen as they are being transferred to the one file to let the user know the system is working.

The only other performance criteria is the size of cell which can be manipulated by the MDCLL system and the size of circuit which can be accommodated. As stated in the requirements (Paragraph 2.4), the memory capacity of the disk being used is the only limiting factor on the size of a final circuit. The size of any given cell, though, is limited by the amount of available memory in the machine, since the cell is stored internally while being worked on. This usable memory is limited to the available memory after the largest program in the MDCLL system is resident along with the operating system. The operating system occupies 16K of memory and, as shown in Table 5.2, the largest program in the MDCLL system occupies 26K of memory. This leaves 20K of memory (with 2K spare) for storing any given cell. The problem now is in determining what can actually be stored in 20K of memory in terms of a cell design. Six cells were encoded in CIF using the MDCLL system to establish the usefulness of the system. The files produced by MDCLL for the six cells are included in Appendix C along with an MCIFPLOT run on each cell. The data resulting from these cells is provided in Table 5.4. Since the rectangle statement provides no great advantage over using CIF, the MDCLL wire statement was used to implement these cells. The six cells implemented are a simple six transistor memory cell, a mask cell for use in a content addressable memory (CAM), a full subtractor, a cell called "BINBOUT" for use in a CAM, which combines the mask, memory and subtractor cells, a cell called "MEM2" which simply calls "BINBOUT" twice, and a cell called MEM8 which iterates eight memory cells. The BINBOUT, MEM2, and MEM8 cells were each FINALIZEd to produce the plots in Appendix C. The statistics shown in Table 5.4 are based on these FINALIZEd files. Another cell, CAMMEM (the entire memory array of a CAM

Cell	1	2	3	4	5	6	7	8
MASK	4	10/0	464	2.3	24	13	85/0	3K
MEM	6	21/0	644	3.2	35	13	97/0	3K
SUB	12	26/0	1080	5.3	40	13	185/0	5K
BINBOUT	22	72/3	493	2.4	131	48	396/3	4K
MEM2	44	72/4	76	0.4	146	48	396/5	1K
MEM8	176	72/4	76	0.4	146	48	396/11	6K

1. Number of transistors contained in cell
2. Number of MDCLL wire statements / calls and iterates used
3. Amount of internal memory used in bytes
4. Percentage of memory used to memory available
5. Number of CIF comment statements in file
6. Number of CIF Layer statements in file
7. Number of CIF Box statements / call statements in file
8. Amount of disk space used in bytes

Table 5.4 Sample MDCLL cell files

chip, 128x16 memory cells) was entered. The CAMMEM cell plot is not included in Appendix C since a cell of this complexity cannot be handled by MCIFPLOT producing a readable plot on one page. The CAMMEM file is not reproduced since the file is identical to the MEM8 file except for 2048 CIF call statements to the BINBOUT cell instead of eight. It took less than two minutes for the FINALIZE program to complete the task of finalizing the CAMMEM and the FINAL.CIF file for this cell occupies 24K of disk space. As shown from the data in Table 5.4 the test cells used less than ten percent of the available memory and still provides for a complex circuit through the use of intermediate cells which call or iterate lower level cells.

6. Recommendations and Conclusions

In this chapter the recommendations for the improvement of the MDCLL system are discussed. Also included are the recommendations for future developments of computer aided design tools in the microcomputer environment. Finally, the conclusions reached, based on the work performed on this thesis, are discussed.

The following recommendations are suggested to improve the usefulness of the MDCLL system:

1. The most important improvement needed to the MDCLL system is to allow the use of normal CIF units (including half lambda measurements). The system must allow the user to enter either integers or floating point numbers and then either store these as integers after a conversion (multiply by four if the system is limited to half lambda measurements) or simply change all internal storage to floating point. This second method would use much more memory space and is therefore not suggested.

2. Changes need to be made to the FINALIZE and DPM programs to ensure that if a standard CIF file has no bounds statement, it can still be instanciated (called or iterated) by an MDCLL cell. Presently, the system errors out if the bounds statement cannot be found. Due to this capability of instanciating existing CIF files, the MDCLL system should not be modified to allow the reading and modification of non-MDCLL created files. This is suggested since the extra code needed to implement the reading and modification of non-MDCLL created CIF files would unnecessarily increase the size and complexity of the MDCLL programs.

3. A desired improvement to the system would be to allow the instanciation of a cell whose description is contained in a library of cells. To do this the DPM program would have to be modified to ask the user the name of the library and then ensure the cell was contained in the library and obtain the bounds statement from the definition of the cell. Also, the FINALIZE program would need to be modified to allow the searching of libraries for the needed cells.

4. In using the MODIFY program, the user should be informed of a change in the cell bounds. This will allow the user to ensure surrounding cells are not affected by the modification or, if needed, can be adjusted accordingly. Since the MDCLL call and iterate statements include the bounds of the cell being instanciated the FINALIZE program should check these bounds against the actual bounds stored with the cell definition to ensure they match. If the bounds don't match the user should be provided with a warning message.

5. The entire INTER program needs to be designed and implemented. This should include the interconnection of cells via the use of the predefined interconnection ports. It would also be desirable to include an automatic router. This ability of using the interconnection ports should also be included in the CREATE and MODIFY programs so the user can give the name of the location to be connected instead of the actual coordinates.

6. The PRIPLO program should be completed. This includes an automatic running of the MCIFPLOT program and the capability to plot the bounding box diagram of any cell in the hierarchy of cells. It would be desirable to implement the bounding box plotting function using only the

standard ASCII character set to ensure compatability with all printers and plotters.

For the future, the MDCLL system should be enhanced, as described above. Also, the following items should be included in the set of computer aided design tools for the microcomputer environment.

1. A program to perform design rule checking on an input CIF file similar to the program DRC presently hosted on the AFIT UNIX/VAX.

2. A circuit extractor program should be implemented to extract the circuit definition based on an input CIF file similar to the MEXTRA program presently hosted on the AFIT UNIX/VAX.

3. A circuit simulator program should be implemented to perform switch level simulations on the circuit described by the file output by the extractor program described above.

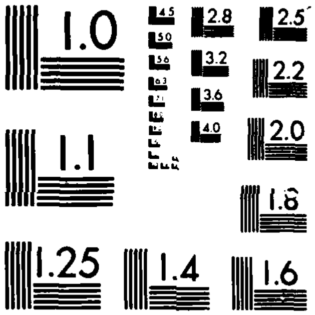
The MDCLL system successfully shows that a high level layout language can be implemented on a microcomputer with limited memory space and still be user friendly. Initially, it was thought the cell being worked on would have to be stored on disk instead of main memory due to the program size. This would have caused noticeable delays to the user while working on a cell. As shown in Paragraph 5.4, the size of the cell which can be implemented using the MDCLL system is quite adequate. This efficient use of memory is obtained by the separation of the functions of the MDCLL system into several independent programs.

Appendix A

Hierarchy/Structure Charts

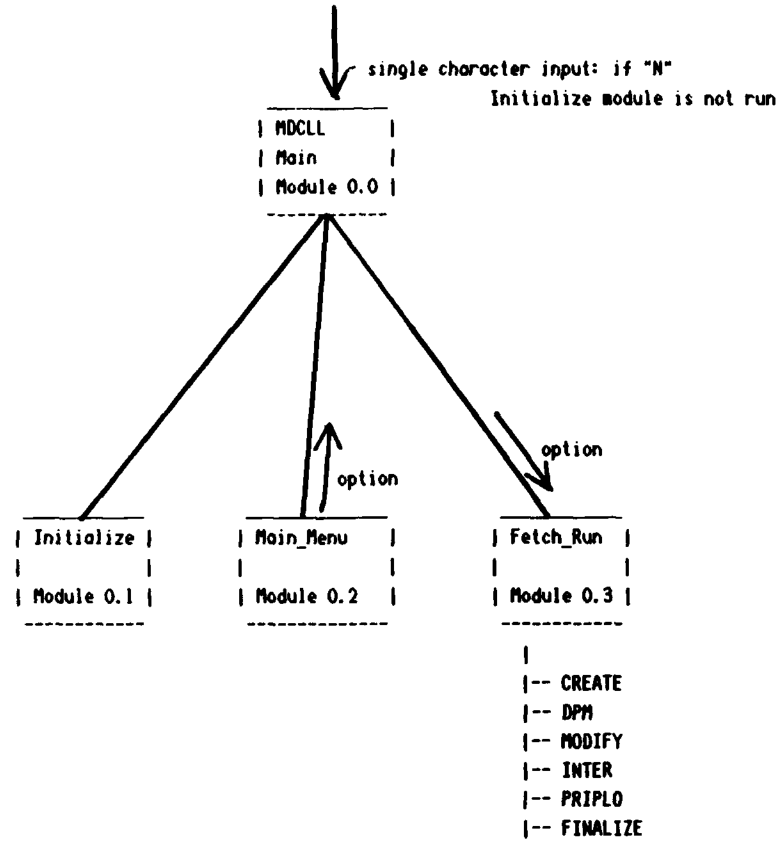
This appendix presents the hierarchy and structure charts for the MDCLL system. They are arranged by program and module number. The hierarchy chart for each program is presented first followed by the structure charts for the modules defined in that program. An asterisk associated with a module in any chart signifies the module is defined under the given module number. An "r" associated with any module signifies the given module is recursive.

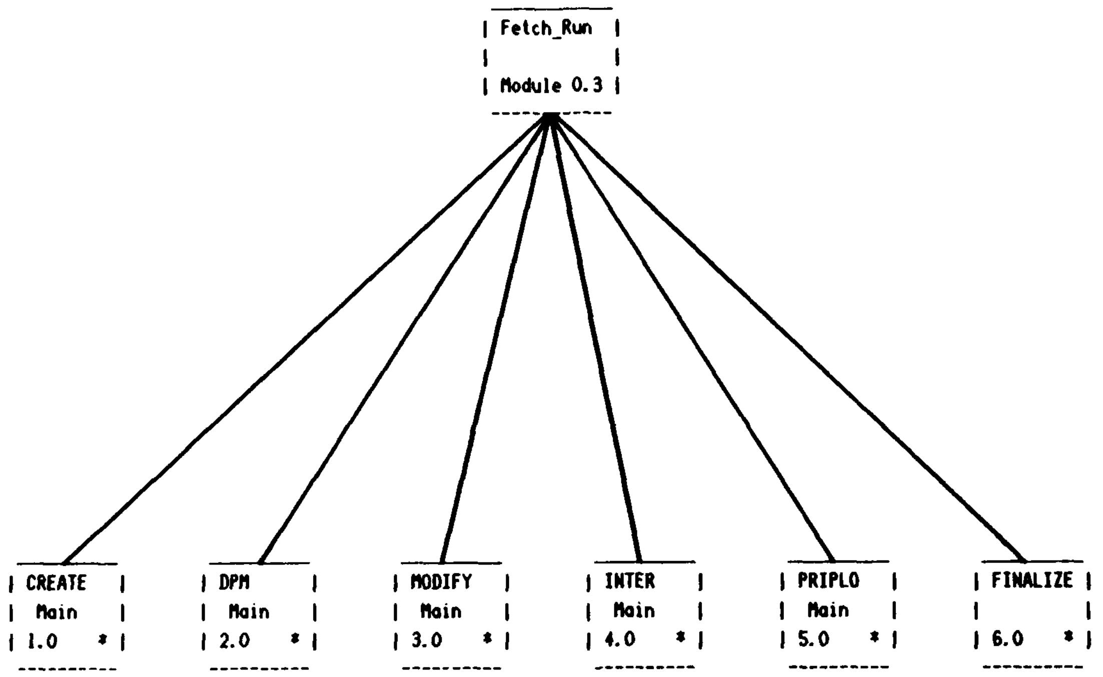
AD-A164 109 A MICROCOMPUTER-BASED MENU-DRIVEN CHIP LAYOUT LANGUAGE 2/4
(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH
SCHOOL OF ENGINEERING S C MORRESE DEC 85
UNCLASSIFIED AFIT/GCS/ENG/85D-11 F/G 9/5 NL



1.0
4.5
5.0
5.6
6.3
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

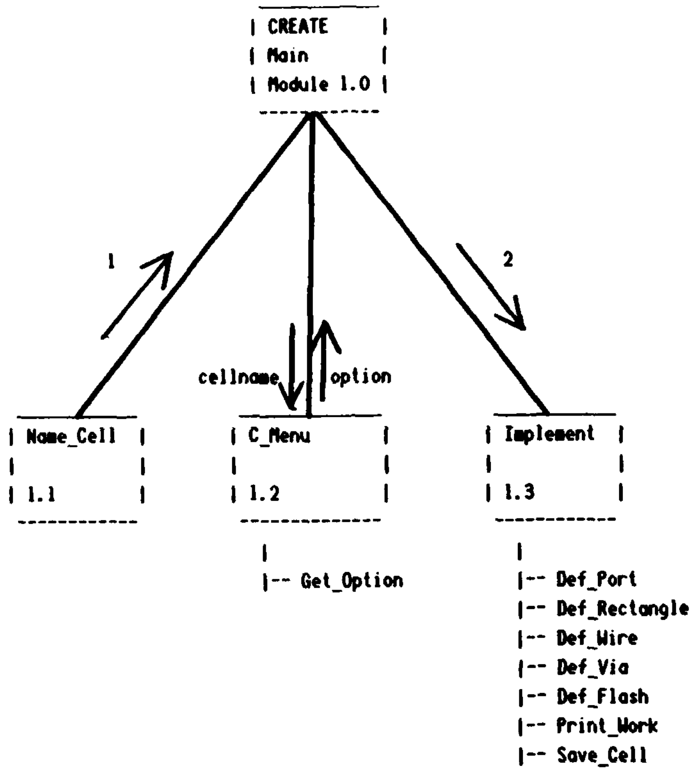
```
*************************************************************************
*                                                                       *
*       MDCLL Program Hierarchy:                                        *
*                                                                       *
*       0.0  MDCLL Main                                                 *
*         |                                                             *
*         |---- 0.1  Initialize                                         *
*         |                                                             *
*         |---- 0.2  Main_Menu                                          *
*         |                                                             *
*         |---- 0.3  Fetch_Run                                          *
*                 |                                                     *
*                 |----  1.0* CREATE Main                               *
*                 |                                                     *
*                 |----  2.0* DPM Main                                  *
*                 |                                                     *
*                 |----  3.0* MODIFY Main                               *
*                 |                                                     *
*                 |----  4.0* INTER Main                                *
*                 |                                                     *
*                 |----  5.0* PRIPLO Main                               *
*                 |                                                     *
*                 |----  6.0* FINALIZE Main                             *
*                                                                       *
*************************************************************************
```


single character input: if "N"
Initialize module is not run
MDCLL
Main
Module 0.0
option
option
Initialize
Module 0.1
Main_Menu
Module 0.2
Fetch_Run
Module 0.3
CREATE
DPM
MODIFY
INTER
PRIPLO
FINALIZE

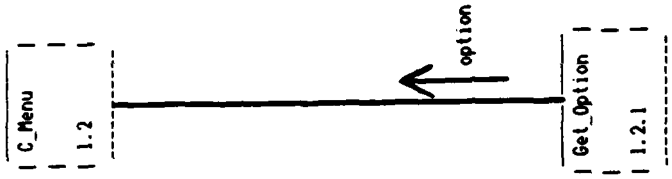


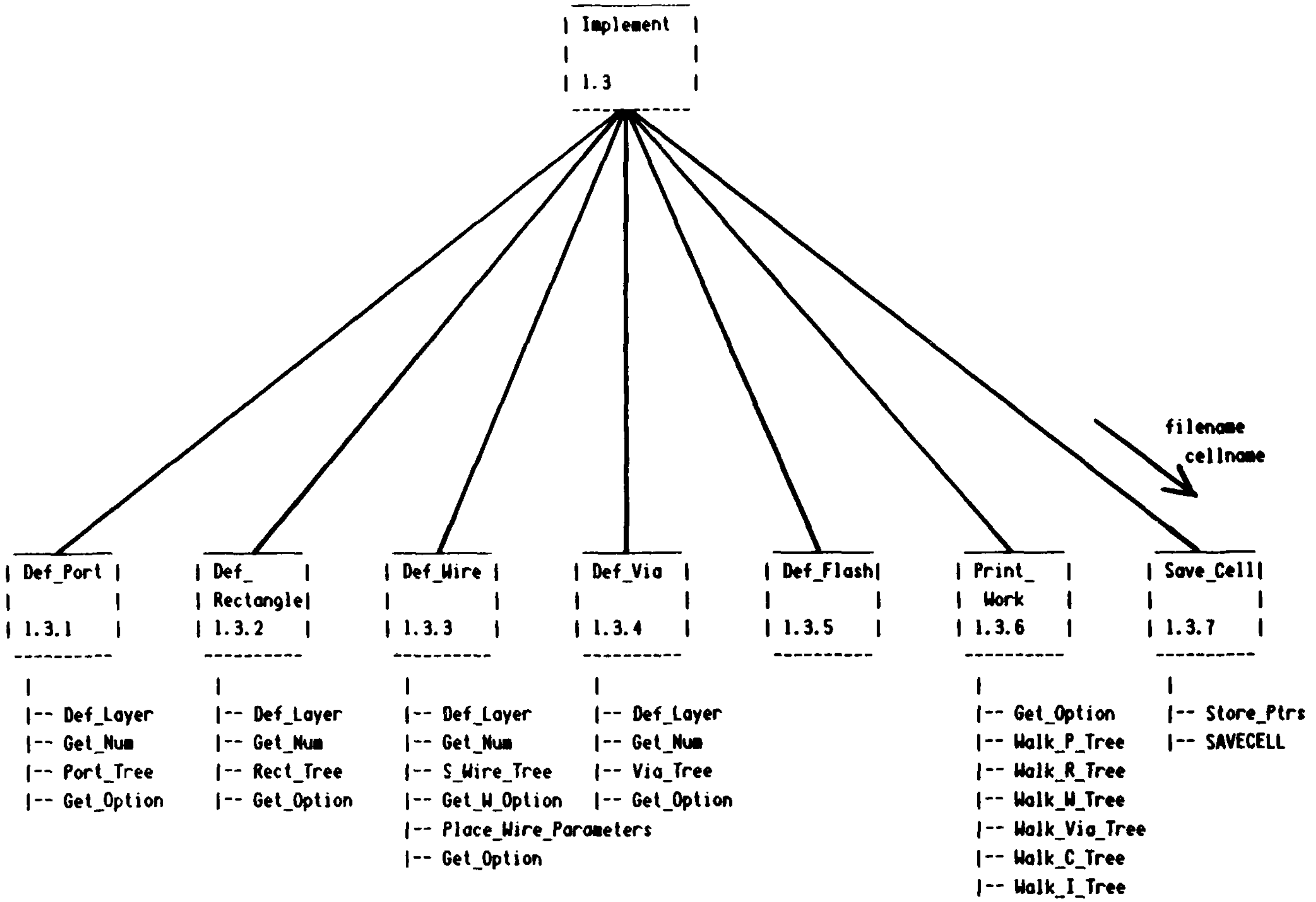
* This module defined under the given module number

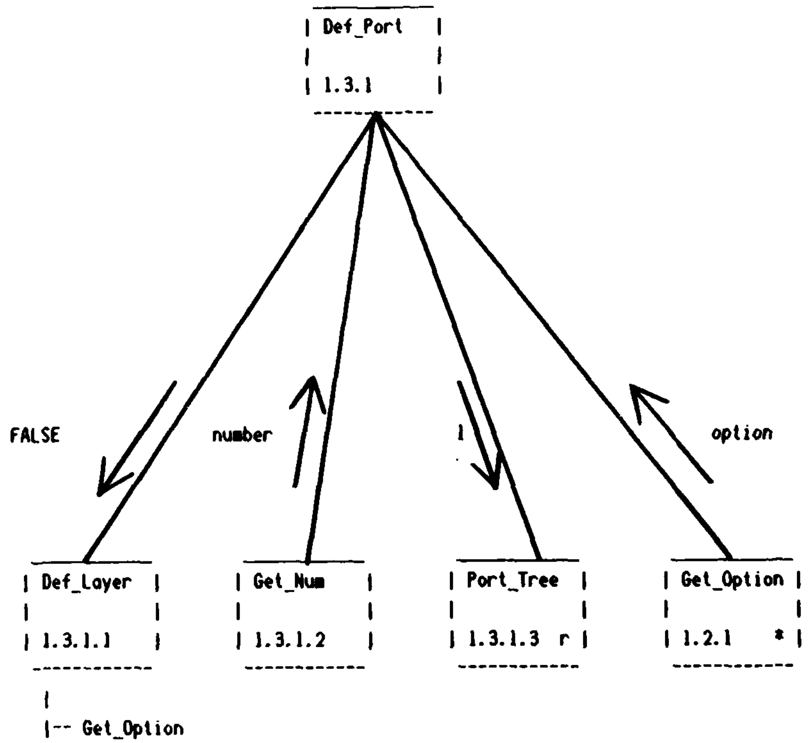
```
*************************************************************************
*       CREATE Program Hierarchy:                                       *
*       1.0  CREATE Main                                                *
*         |                                                             *
*         |---- 1.1  Name_cell                                          *
*         |                                                             *
*         |---- 1.2  C_Menu                                             *
*         |       |---- 1.2.1  Get_Option                               *
*         |                                                             *
*         |---- 1.3  Implement                                          *
*                 |                                                     *
*                 |---- 1.3.1  Def_Port                                 *
*                 |         |---- 1.3.1.1  Def_Layer                    *
*                 |         |---- 1.3.1.2  Get_Num                      *
*                 |         |---- 1.3.1.3r Port_Tree                    *
*                 |         |---- 1.2.1* Get_Option                     *
*                 |                                                     *
*                 |---- 1.3.2  Def_Rectangle                            *
*                 |         |---- 1.3.1.1* Def_Layer                    *
*                 |         |---- 1.3.1.2* Get_Num                      *
*                 |         |---- 1.3.2.1r Rect_Tree                    *
*                 |         |---- 1.2.1* Get_Option                     *
*                 |                                                     *
*                 |---- 1.3.3  Def_Wire                                 *
*                 |         |---- 1.3.1.1* Def_Layer                    *
*                 |         |---- 1.3.1.2* Get_Num                      *
*                 |         |---- 1.3.3.1r S_Wire_Tree                  *
*                 |         |---- 1.3.3.2  Get_W_Option                 *
*                 |         |---- 1.3.3.3r Place_Wire_Parameters        *
*                 |         |---- 1.2.1* Get_Option                     *
*                 |                                                     *
*                 |---- 1.3.4  Def_Via                                  *
*                 |         |---- 1.3.1.1* Def_Layer                    *
*                 |         |---- 1.3.1.2* Get_Num                      *
*                 |         |---- 1.3.4.1r Via_Tree                     *
*                 |         |---- 1.2.1* Get_Option                     *
*                 |                                                     *
*                 |---- 1.3.5  Def_Flash                                *
*                 |                                                     *
*                 |---- 1.3.6  Print_Work                               *
*                 |         |---- 1.2.1* Get_Option                     *
*                 |         |---- 1.3.6.1r Walk_P_Tree                  *
*                 |         |---- 1.3.6.2r Walk_R_Tree                  *
*                 |         |---- 1.3.6.3r Walk_W_Tree                  *
*                 |         |           |---- 1.3.6.3.1r Segment_Tree   *
*                 |         |---- 1.3.6.4r Walk_Via_Tree                *
*                 |         |---- 1.3.6.5r Walk_C_Tree                  *
*                 |         |---- 1.3.6.6r Walk_I_Tree                  *
*                 |---- 1.3.7  Save_Cell                                *
*                           |---- 1.3.7.1  Store_Ptrs                   *
*                           |---- 7.0  SAVECELL Main                    *
*************************************************************************
```



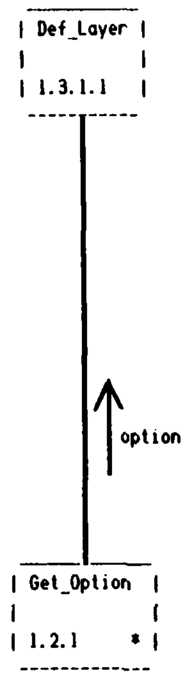
1. on_disk, filename, cellname
2. option, filename, cellname


C_Menu
1.2
option
Get_Option
1.2.1

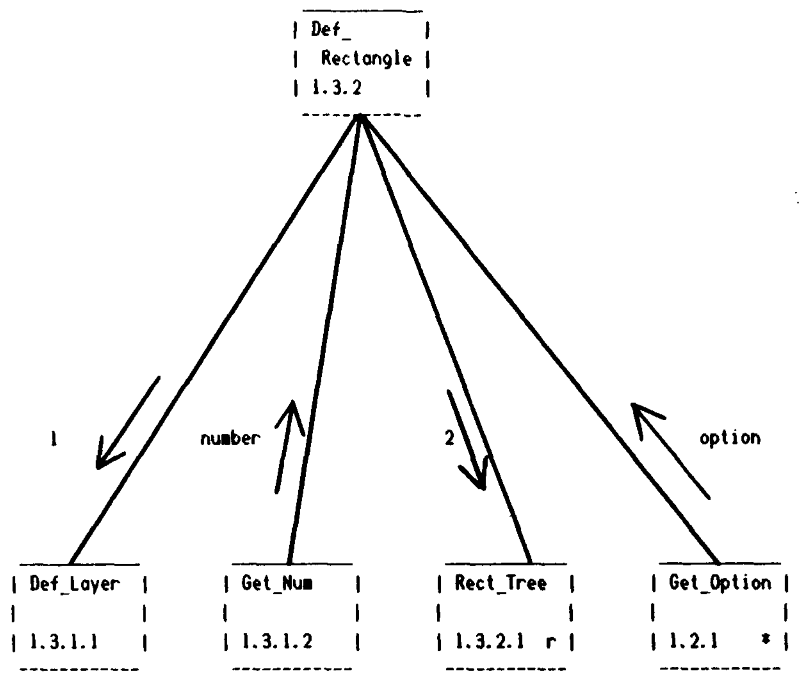

Implement
1.3
filename
cellname
Def_Port
1.3.1
Def_Layer
Get_Num
Port_Tree
Get_Option
Def_
Rectangle
1.3.2
Def_Layer
Get_Num
Rect_Tree
Get_Option
Def_Wire
1.3.3
Def_Layer
Get_Num
S_Wire_Tree
Get_W_Option
Place_Wire_Parameters
Get_Option
Def_Via
1.3.4
Def_Layer
Get_Num
Via_Tree
Get_Option
Def_Flash
1.3.5
Print_
Work
1.3.6
Get_Option
Walk_P_Tree
Walk_R_Tree
Walk_W_Tree
Walk_Via_Tree
Walk_C_Tree
Walk_I_Tree
Save_Cell
1.3.7
Store_Ptrs
SAVECELL



* This module defined under the given module number
r This module is recursive
1. portname, p_x, p_y, port_ptr



* This module defined under the given module number

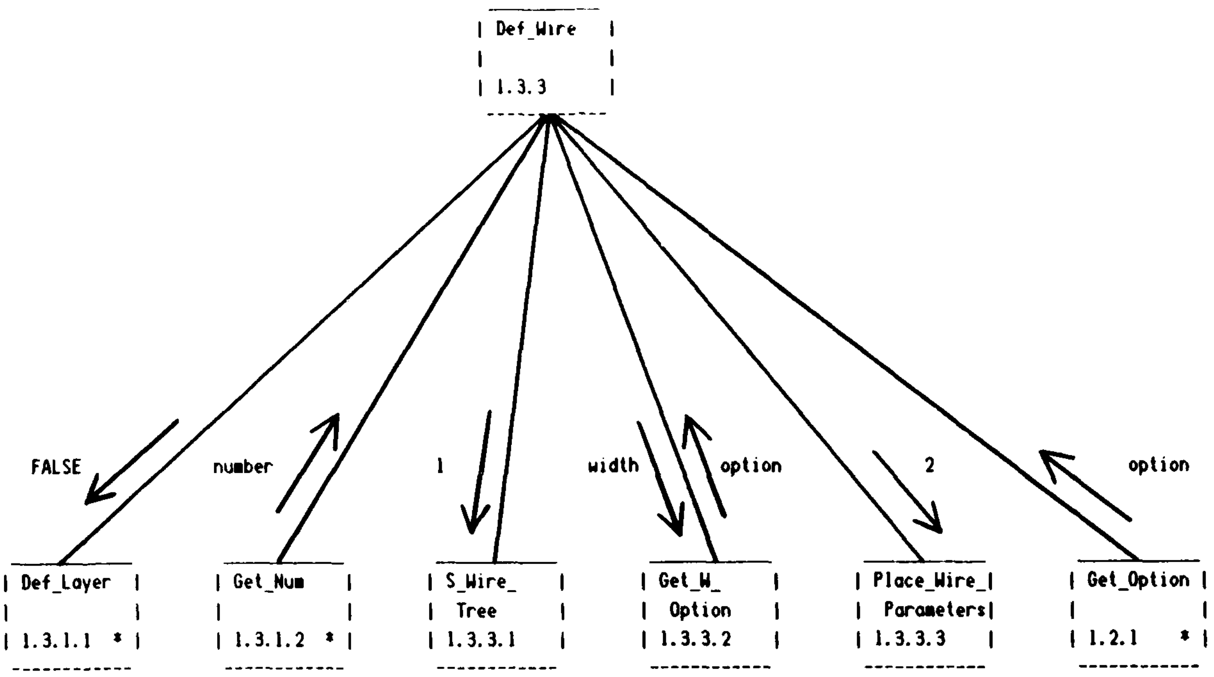


* This module defined under the given module number

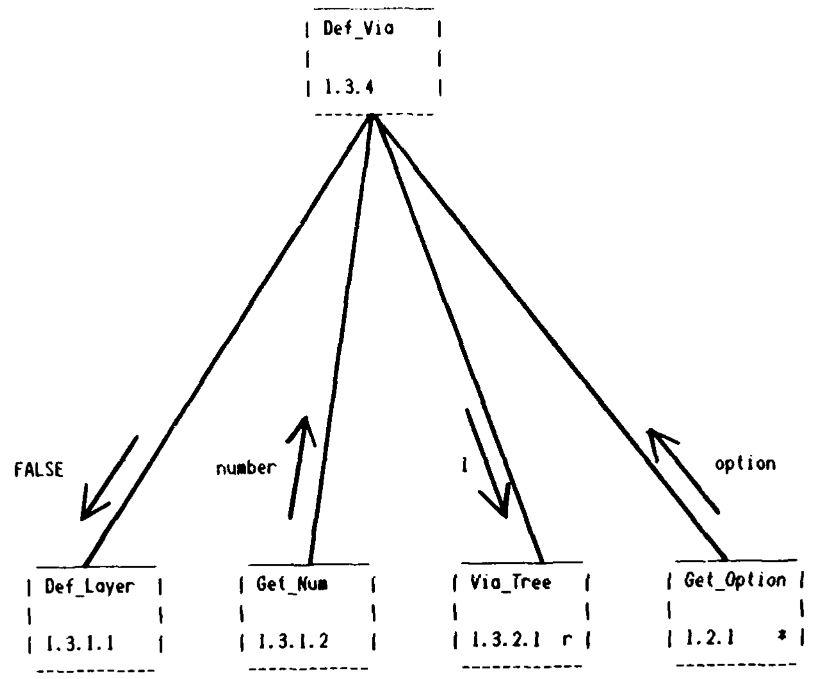
r This module is recursive

1. TRUE or FALSE

2. r_h, r_l, r_x, r_y, ptr to rectangle (s_rect,d_rect,p_rect, or f_rect)



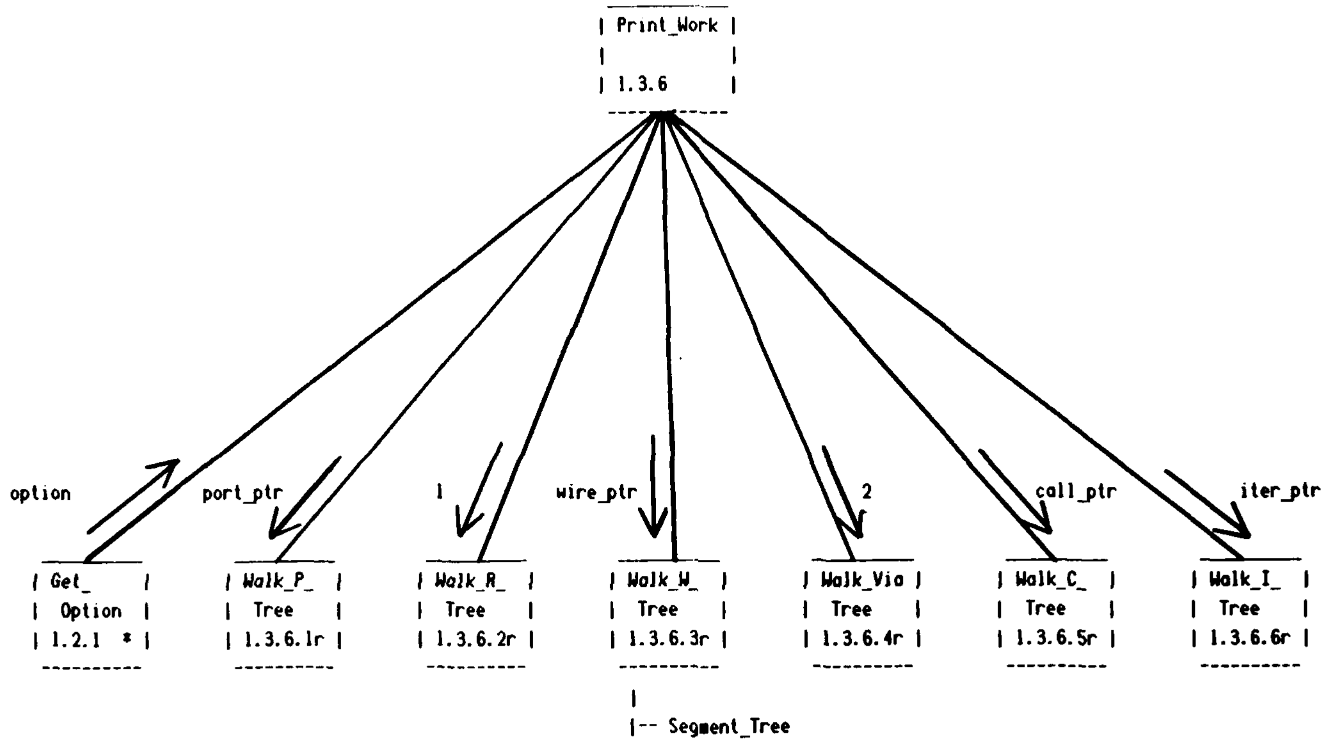
* This module defined under the given module number
r This module is recursive
1. w_x, w_y, wire_ptr
2. width, length, option, seg_ptr



* This module defined under the given module number

r This module is recursive

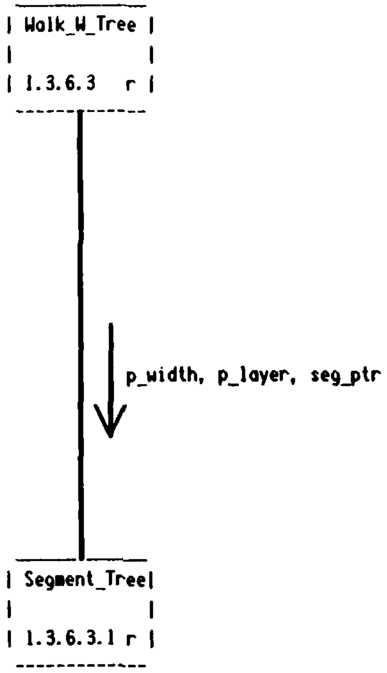
1. x_loction, y_loction, prt to via structure (s_via, d_via, p_via)



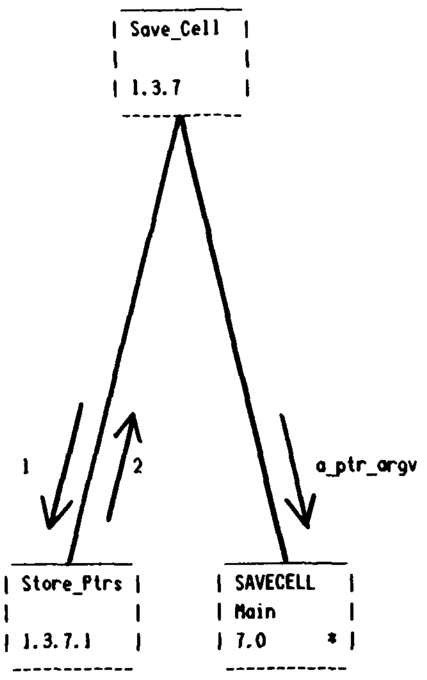
* This module defined under the given module number

r This module is recursive

1. ptr to rectangle (f_rect, s_rect, d_rect, p_rect)
2. ptr to via (s_via, d_via, p_via)



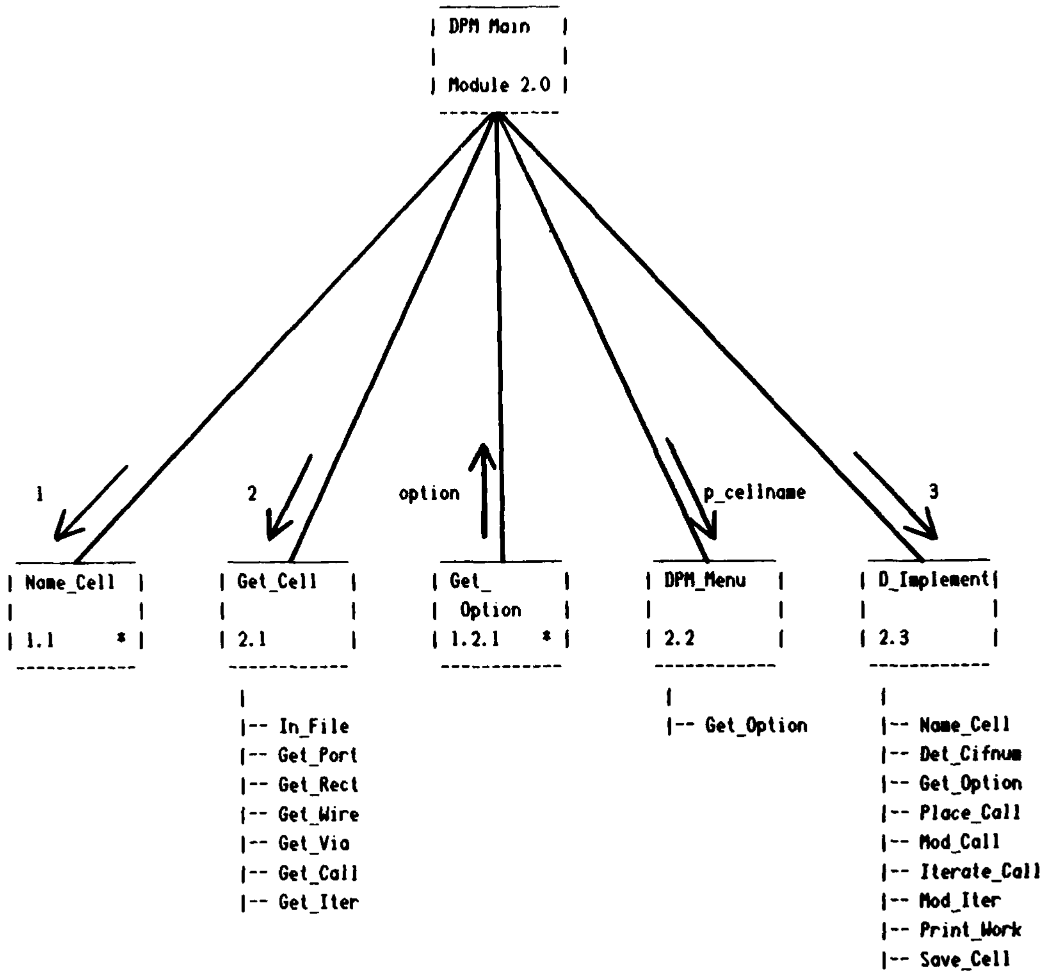
r This module is recursive



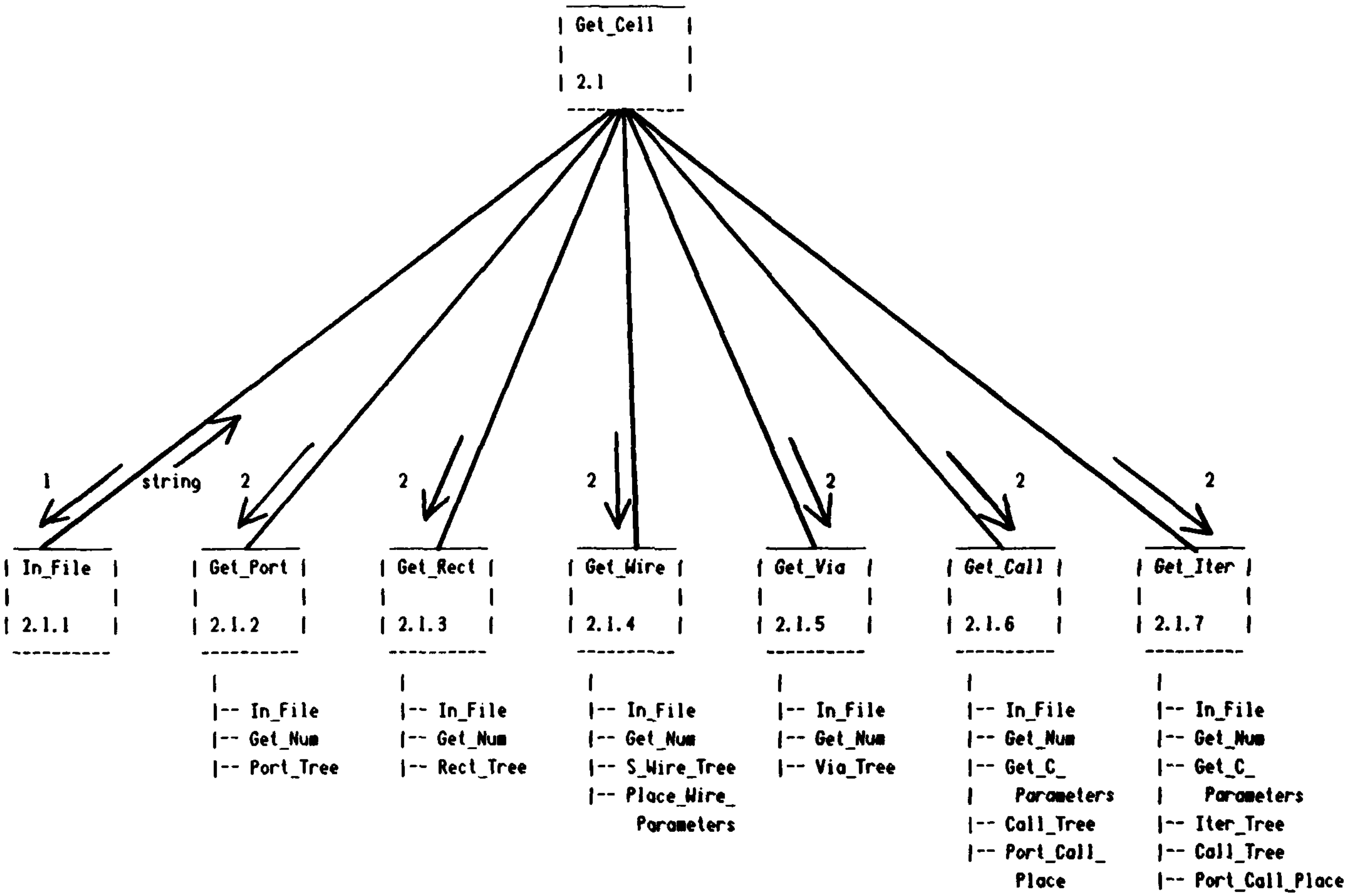
* This module defined under the given module number

1. a_ptr_argv, i, ptr to structure (port_ptr, f_rect, s_rect, d_rect, p_rect, s_via, p_via, d_via, wire_ptr, call_ptr, iter_ptr)
2. updated a_ptr_argv, i

```
*************************************************************************
*                                                                       *
*       DPM Program Hierarchy:                                          *
*                                                                       *
*                                                                       *
*       2.0  DPM Main                                                   *
*         |                                                             *
*         |---- 1.1* Name_Cell                                          *
*         |                                                             *
*         |---- 2.1  Get_Cell                                           *
*         |       |                                                     *
*         |       |---- 2.1.1  In_File                                  *
*         |       |                                                     *
*         |       |---- 2.1.2  Get_Port                                 *
*         |       |         |---- 2.1.1* In_File                        *
*         |       |         |---- 1.3.1.2* Get_Num                      *
*         |       |         |---- 1.3.1.3* Port_Tree                    *
*         |       |                                                     *
*         |       |---- 2.1.3  Get_Rect                                 *
*         |       |         |---- 2.1.1* In_File                        *
*         |       |         |---- 1.3.1.2* Get_Num                      *
*         |       |         |---- 1.3.2.1* Rect_Tree                    *
*         |       |                                                     *
*         |       |---- 2.1.4  Get_Wire                                 *
*         |       |         |---- 2.1.1* In_File                        *
*         |       |         |---- 1.3.1.2* Get_Num                      *
*         |       |         |---- 1.3.3.1* S_Wire_Tree                  *
*         |       |         |---- 1.3.3.3* Place_Wire_Parameters        *
*         |       |                                                     *
*         |       |---- 2.1.5  Get_Via                                  *
*         |       |         |---- 2.1.1* In_File                        *
*         |       |         |---- 1.3.1.2* Get_Num                      *
*         |       |         |---- 1.3.4.1* Via_Tree                     *
*         |       |                                                     *
*         |       |---- 2.1.6  Get_Call                                 *
*         |       |         |---- 2.1.1* In_File                        *
*         |       |         |---- 1.3.1.2* Get_Num                      *
*         |       |         |---- 2.1.6.1  Get_C_Parameters             *
*         |       |         |           |---- 1.3.1.2* Get_Num          *
*         |       |         |---- 2.3.1.2* Call_Tree                    *
*         |       |         |---- 2.3.2.3.1* Port_Call_Place            *
*         |       |                                                     *
*         |       |---- 2.1.7  Get_Iter                                 *
*         |                 |---- 2.1.1* In_File                        *
*         |                 |---- 1.3.1.2* Get_Num                      *
*         |                 |---- 2.1.6.1* Get_C_Parameters             *
*         |                 |---- 2.3.3.2* Iter_Tree                    *
*         |                 |---- 2.3.1.2* Call_Tree                    *
*         |                 |---- 2.3.1.3.1* Port_Call_Place            *
*         |                                                             *
*         |                                                              *
*         |---- 1.2.1* Get_Option                                        *
*         |                                                              *
*         |---- 2.2  DPM_Menu                                            *
*         |       |---- 1.2.1* Get_Option                                *
*         |                                                              *
*         |---- 2.3  D_Implement                                         *
*                 |                                                      *
*                 |---- 1.1* Name_Cell                                   *
*                 |                                                      *
*                 |---- 7.1* Det_Cifnum                                  *
*                 |                                                      *
*                 |---- 1.2.1* Get_Option                                *
*                 |                                                      *
*                 |---- 2.3.1  Place_Call                                *
*                 |         |---- 2.3.1.1  Call_Parameters               *
*                 |         |            |---- 1.3.1.2* Get_Num          *
*                 |         |            |---- 1.2.1* Get_Option         *
*                 |         |---- 2.3.1.2r Call_Tree                     *
*                 |         |---- 2.3.1.3  C_Port_Names                  *
*                 |         |            |---- 1.2.1* Get_Option         *
*                 |         |            |---- 2.3.1.3.1r Port_Call_Place*
*                 |         |---- 1.2.1* Get_Option                      *
*                 |                                                      *
*                 |---- 2.3.2r Mod_Call                                  *
*                 |         |---- 1.2.1* Get_Option                      *
*                 |         |---- 2.3.1.1* Call_Parameters               *
*                 |         |---- 2.3.1.3* C_Port_Names                  *
*                 |                                                      *
*                 |---- 2.3.3  Iterate_Call                              *
*                 |         |---- 2.3.1.1* Call_Parameters               *
*                 |         |---- 2.3.3.1  Iter_Parameters               *
*                 |         |            |---- 1.3.1.3* Get_Num          *
*                 |         |---- 2.3.3.2r Iter_Tree                     *
*                 |         |---- 2.3.1.2* Call_Tree                     *
*                 |         |---- 2.3.1.3* C_Port_Names                  *
*                 |                                                      *
*                 |---- 2.3.4r Mod_Iter                                  *
*                 |         |---- 1.2.1* Get_Option                      *
*                 |         |---- 2.3.3.1* Iter_Parameters               *
*                 |         |---- 2.3.1.1* Call_Parameters               *
*                 |         |---- 2.3.1.3* C_Port_Names                  *
*                 |                                                      *
*                 |---- 1.3.6* Print_Work                                *
*                 |                                                      *
*                 |---- 1.3.7* Save_Cell                                 *
*                                                                        *
**************************************************************************
```



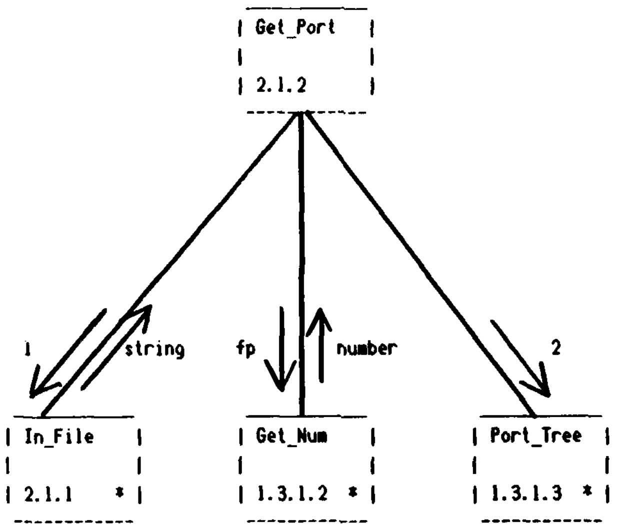
* This module defined under the given module number
r This module is recursive
1. on_disk, p_filename, p_cellname
2. p_filename, p_cellname
3. option, p_filename, p_cellname



* This module defined under the given module number

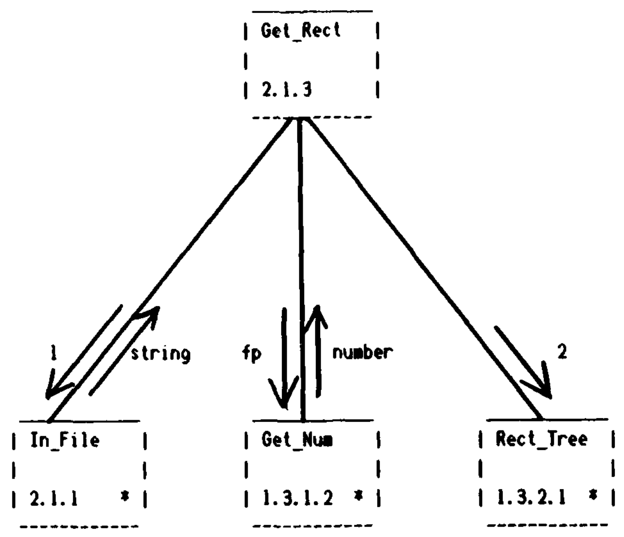
r This module is recursive

1. fp, string, character being searched for
2. fp, string



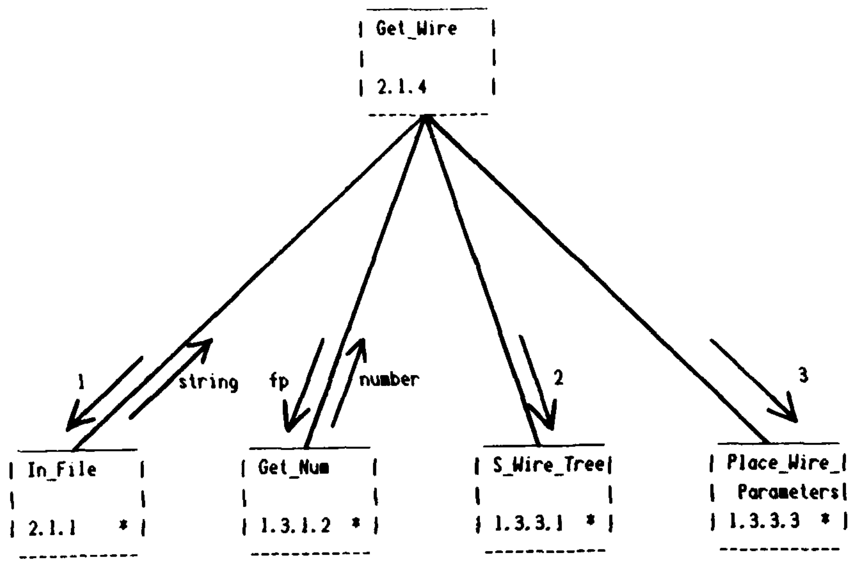
* This module defined under the given module number

1. fp, string, character being searched for
2. portname, p_x, p_y, port_ptr



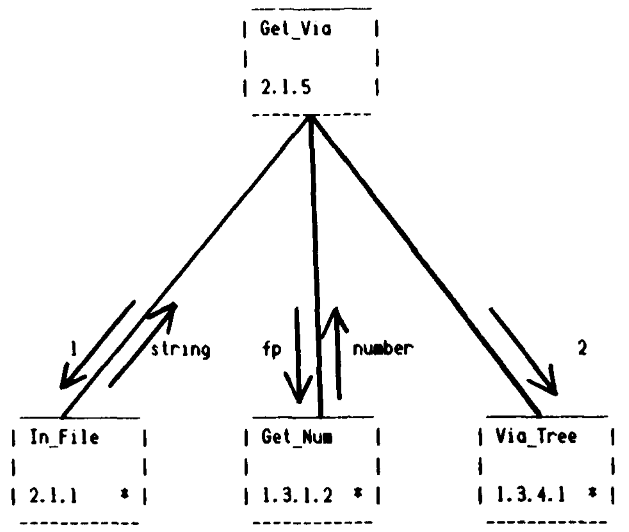
* This module defined under the given module number

1. fp, string, character being searched for
2. r_h, r_l, r_x, r_y, ptr to proper layer of
   rectangles (f_rect, s_rect, p_rect, d_rect)



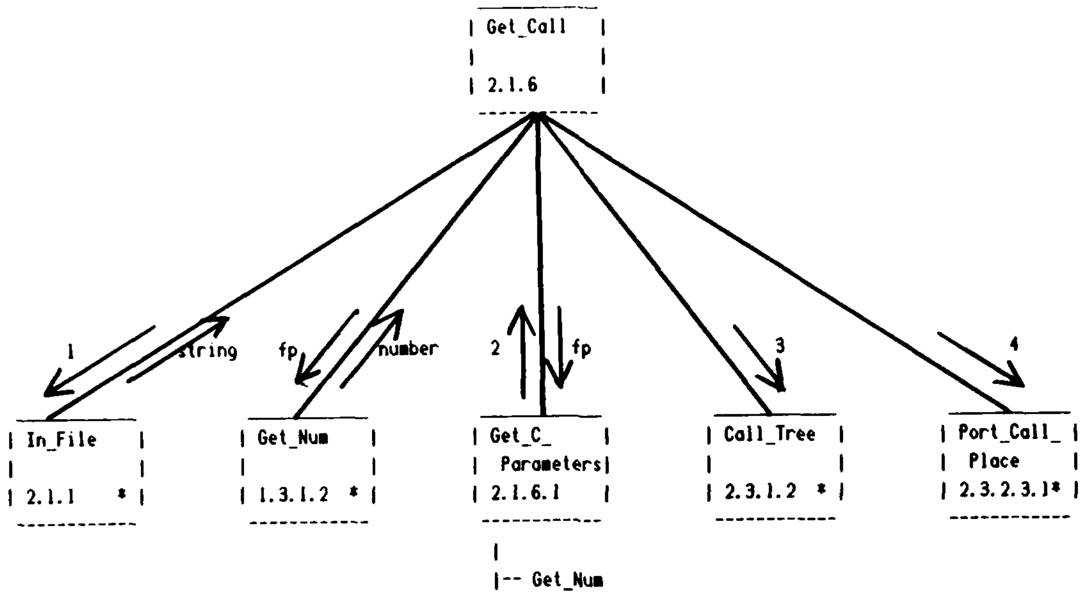
* This module defined under the given module number

1. fp, string, character being searched for
2. w_x, w_y, wire_ptr
3. width, length, direction, seg_ptr



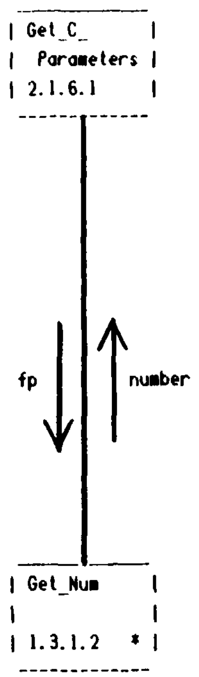
* This module defined under the given module number

1. fp, string, character to be searched for
2. v_x, v_y, ptr to proper layer of vias (s_via, p_via, d_via)

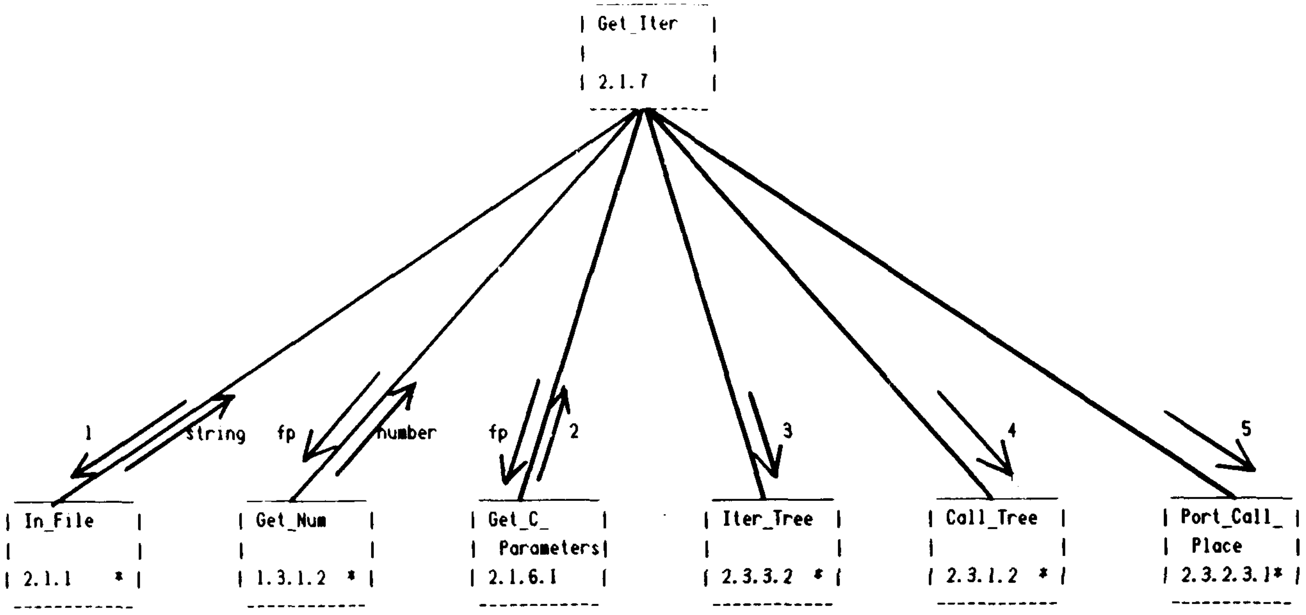


* This module defined under the given module number

1. fp, string, character being searched for
2. cifnum, m_x, m_y, r_x
3. cellname, cifnum, t_x, t_y, m_x, m_y, r_x,
   min_x, min_y, max_x, max_y
4. portname, c_port_ptr

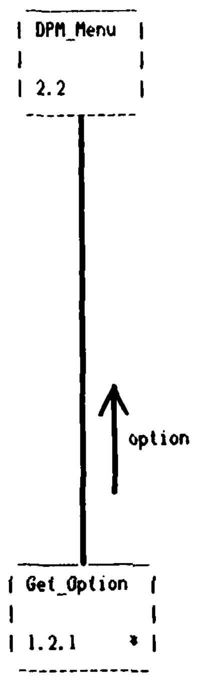


* This module defined under the given module number

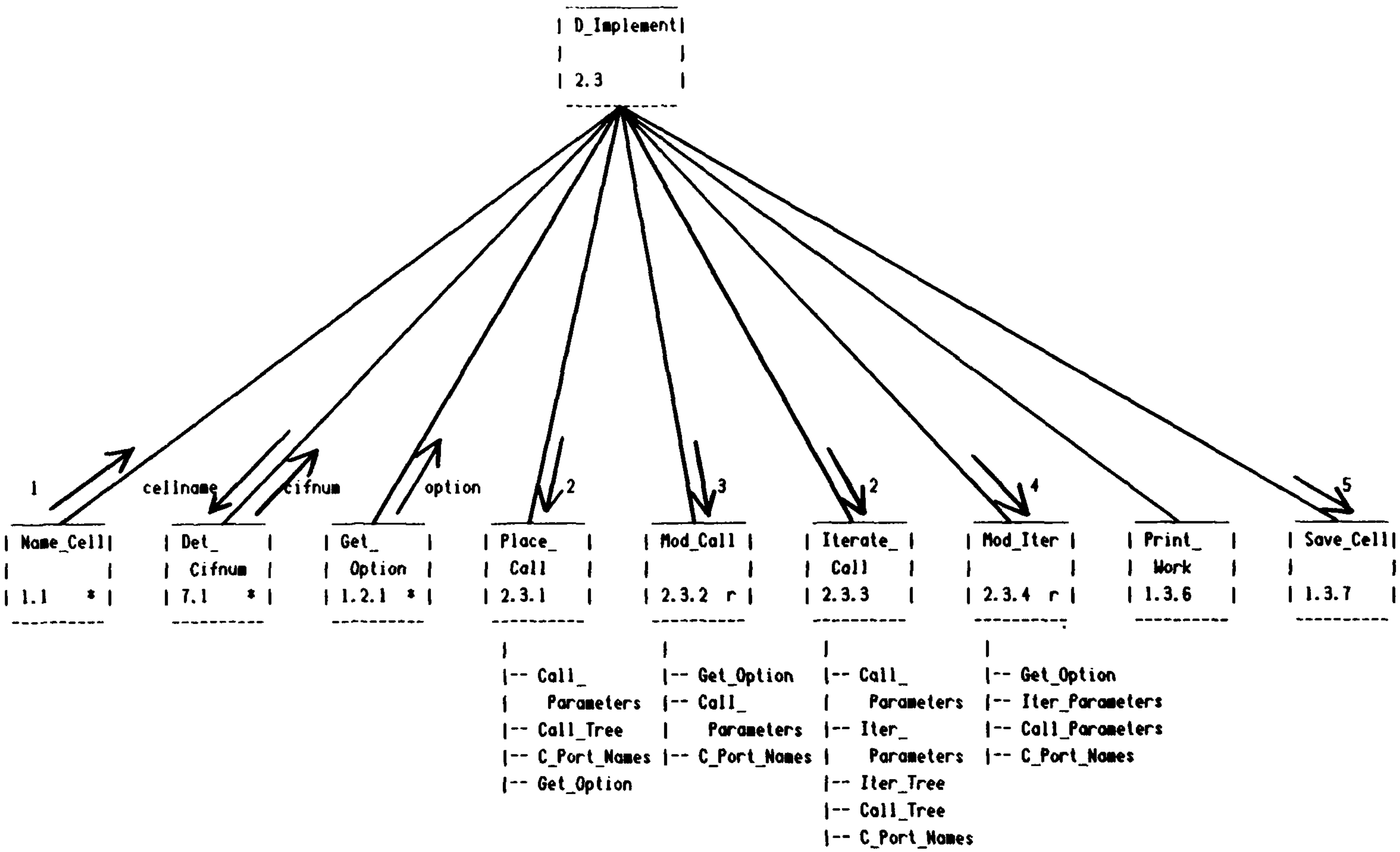


* This module defined under the given module number

1. fp, string, character being searched for
2. cifnum, m_x, m_y, r_x
3. cellname, cifnum, n_x, n_y, x_p, y_p, iter_ptr
4. cellname, cifnum, t_x, t_y, m_x, m_y, r_x,
   min_x, min_y, max_x, max_y, i_call_ptr
5. portname, c_port_ptr



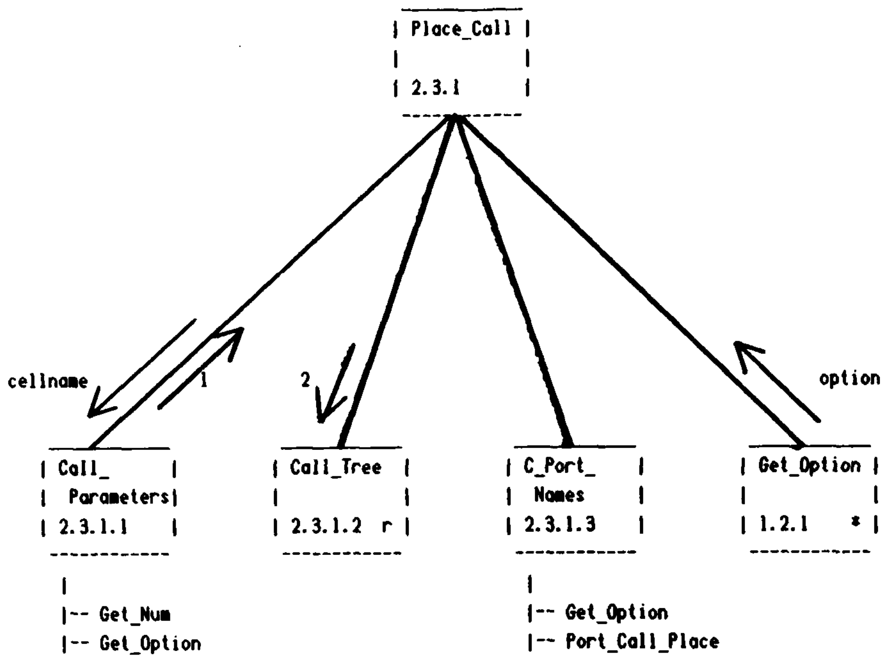
* This module defined under the given module number



* This module defined under the given module number

r This module is recursive

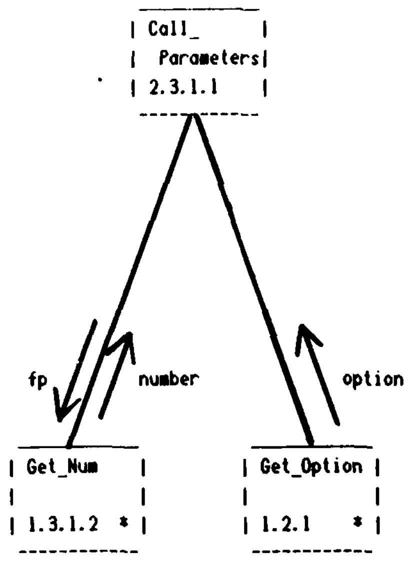
1. on_disk, filename, cellname
2. min_x, min_y, max_x, max_y, cifnum, cellname
3. exit, cifnum, call_ptr
4. exit, cifnum, iter_ptr
5. p_filename, p_cellname



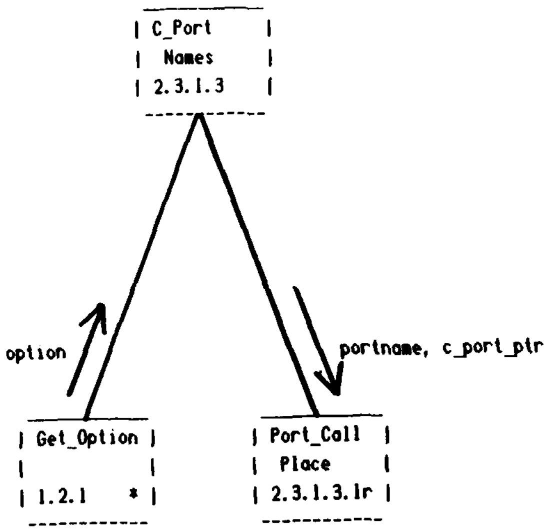
* This module defined under the given module number

r This module is recursive

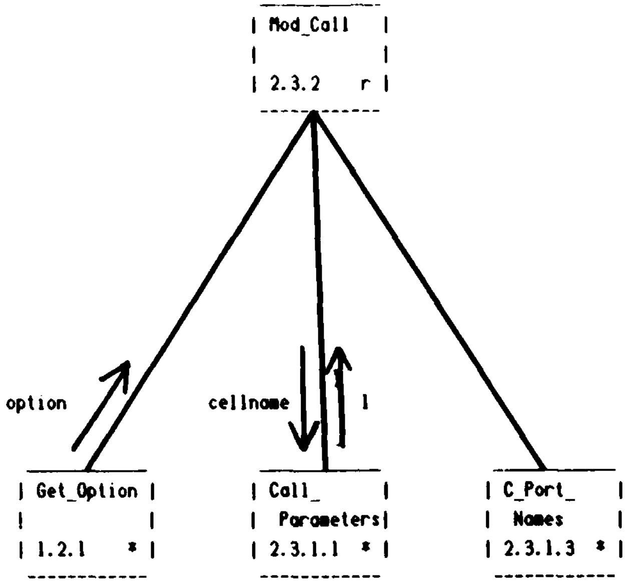
1. t_x, t_y, m_x, m_y, r_x
2. cellname, cifnum, t_x, t_y, m_x, m_y, r_x,
   min_x, min_y, max_x, max_y, call_ptr



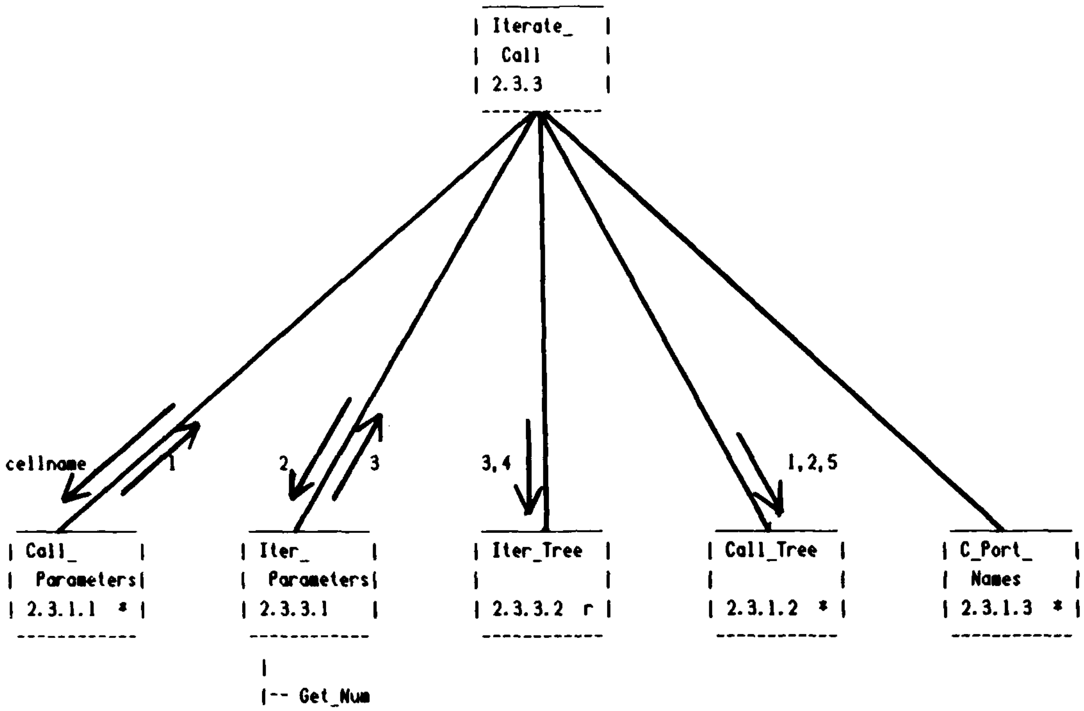
* This module defined under the given module number



* This module defined under the given module number



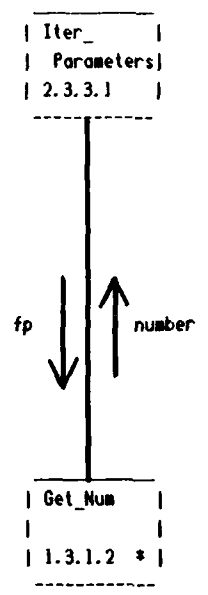
* This module defined under the given module number
r This module is recursive
1. t_x, t_y, m_x, m_y, r_x



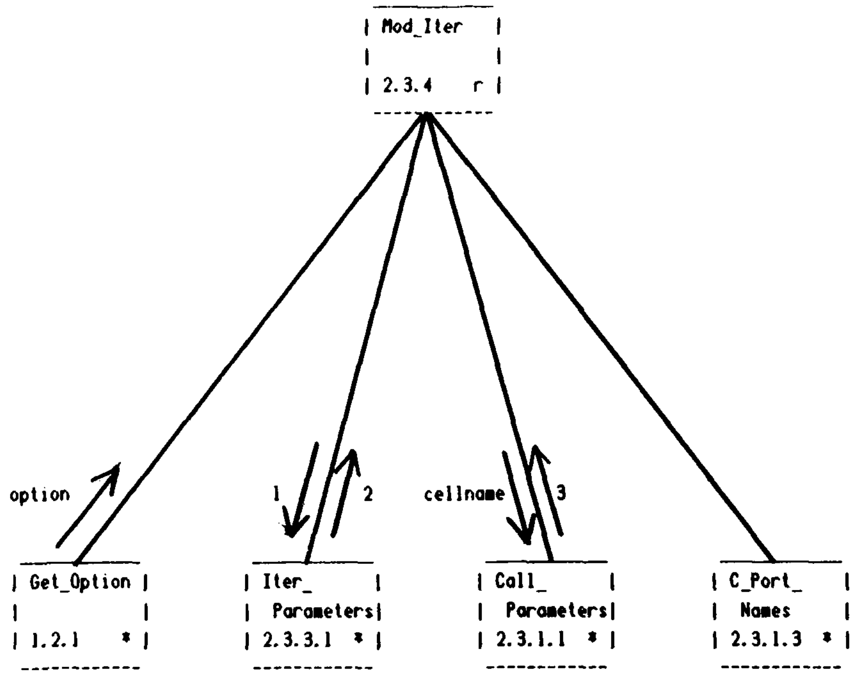
* This module defined under the given module number

r This module is recursive

1. t_x, t_y, m_x, m_y, r_x
2. min_x. min_y, max_x, max_y
3. n_x, n_y, x_p, y_p
4. cellname, cifnum, iter_ptr
5. cellname, cifnum, i_call_ptr

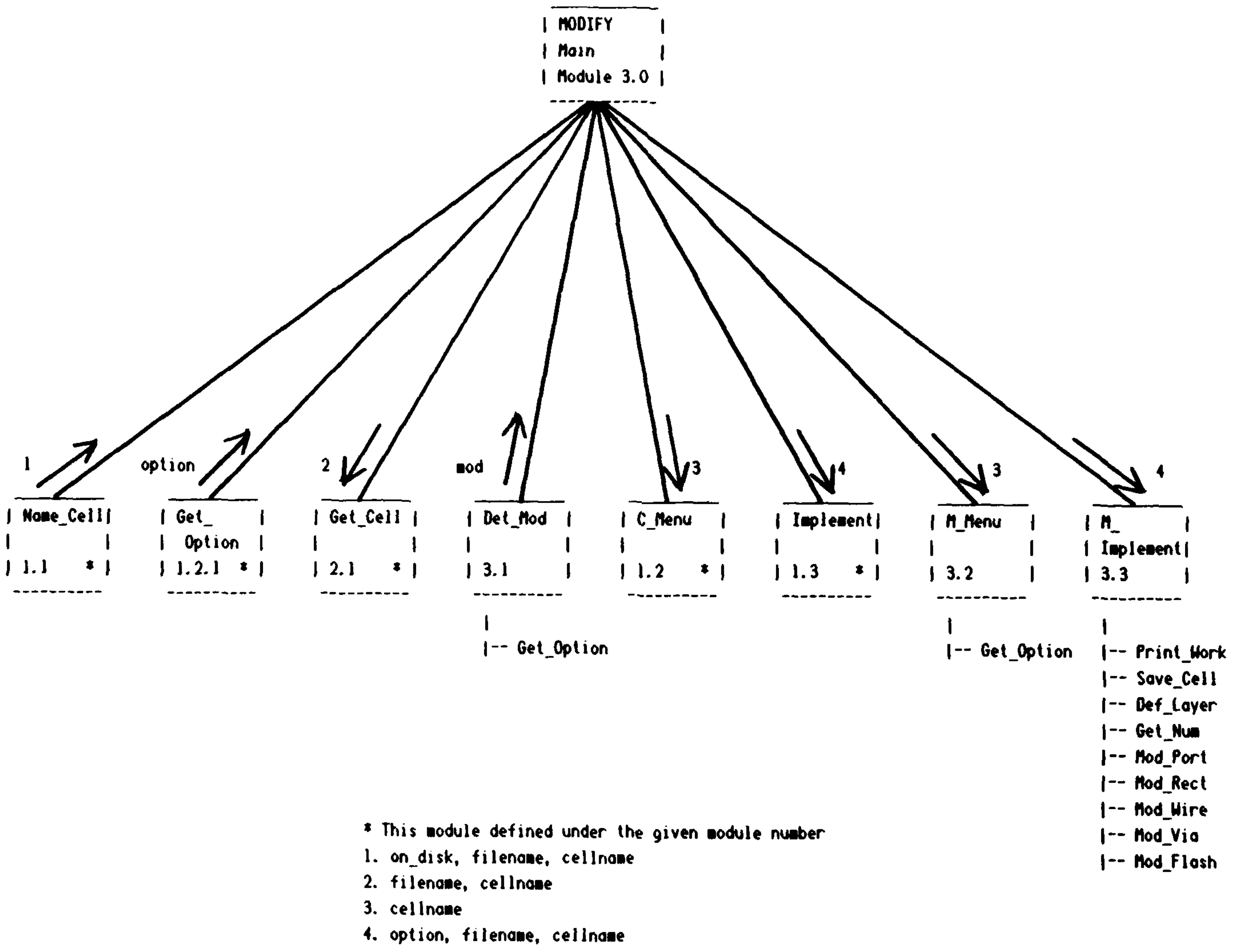


* This module defined under the given module number



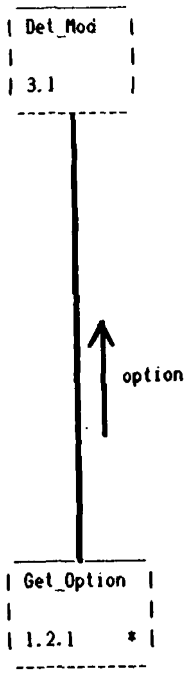
* This module defined under the given module number
r This module is recursive
1. x_min, y_min, x_max, y_max
2. n_x, n_y, x_p, y_p
3. t_x, t_y, m_x, m_y, r_x

```
**************************************************************************
*       MODIFY Program Hierarchy:                                        *
*       3.0  MODIFY Main                                                 *
*         |                                                              *
*         |---- 1.1* Name_Cell                                           *
*         |                                                              *
*         |---- 1.2.1* Get_Option                                        *
*         |                                                              *
*         |---- 2.1* Get_Cell                                            *
*         |                                                              *
*         |---- 3.1  Det_Mod                                             *
*         |       |---- 1.2.1* Get_Option                                *
*         |                                                              *
*         |---- 1.2* C_Menu                                              *
*         |                                                              *
*         |---- 1.3* Implement                                           *
*         |                                                              *
*         |---- 3.2  M_Menu                                              *
*         |       |---- 1.2.1* Get_Option                                *
*         |                                                              *
*         |---- 3.3 M_Implement                                          *
*                 |                                                      *
*                 |---- 1.3.6* Print_Work                                *
*                 |                                                      *
*                 |---- 1.3.7* Save_Cell                                 *
*                 |                                                      *
*                 |---- 1.3.1.1* Def_Layer                               *
*                 |                                                      *
*                 |---- 1.3.1.2* Get_Num                                 *
*                 |                                                      *
*                 |---- 3.3.1r Mod_Port                                  *
*                 |         |---- 1.2.1* Get_Option                      *
*                 |         |---- 1.3.1.1* Def_Layer                     *
*                 |         |---- 1.3.1.2* Get_Num                       *
*                 |                                                      *
*                 |---- 3.3.2r Mod_Rect                                  *
*                 |         |---- 1.2.1* Get_Option                      *
*                 |         |---- 1.3.1.2* Get_Num                       *
*                 |         |---- 1.3.2.1* Rect_Tree                     *
*                 |                                                      *
*                 |---- 3.3.3r Mod_Wire                                  *
*                 |         |---- 1.3.6.3.1* Segment_Tree                *
*                 |         |---- 1.2.1* Get_Option                      *
*                 |         |---- 1.3.1.2* Get_Num                       *
*                 |         |---- 1.3.3.1* S_Wire_Tree                   *
*                 |         |---- 1.3.3.3* Place_Wire_Parameters         *
*                 |---- 3.3.4r Mod_Via                                   *
*                 |         |---- 1.2.1* Get_Option                      *
*                 |         |---- 1.3.1.2* Get_Num                       *
*                 |         |---- 1.3.4.1* Via_Tree                      *
*                 |---- 3.3.5  Mod_Flash                                 *
**************************************************************************
```

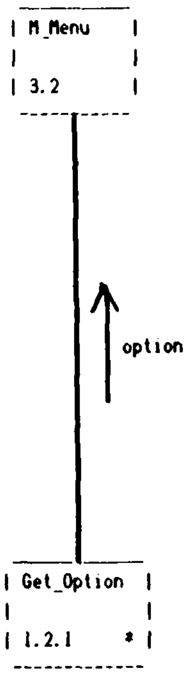


* This module defined under the given module number

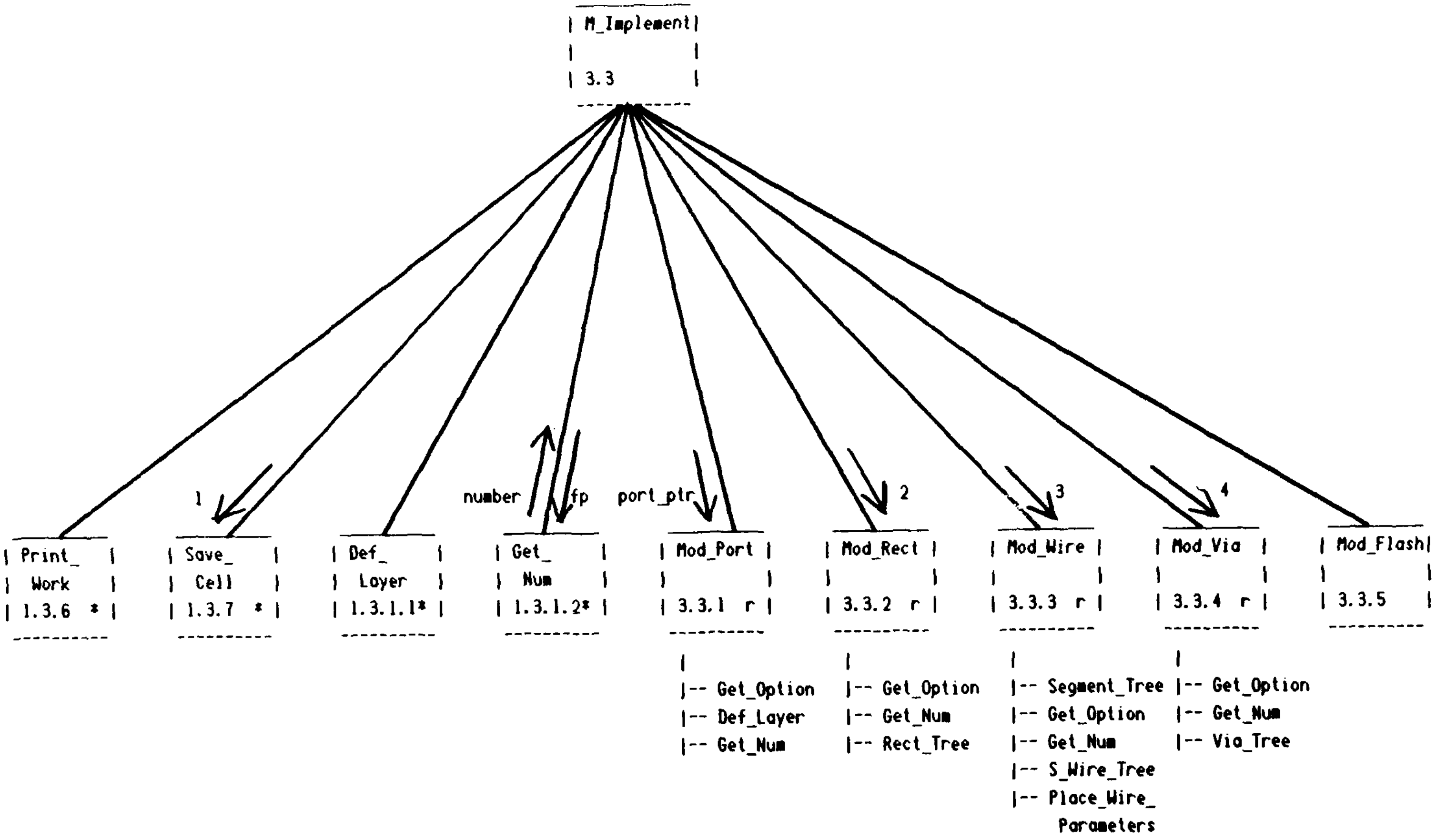
1. on_disk, filename, cellname
2. filename, cellname
3. cellname
4. option, filename, cellname



* This module defined under the given module number



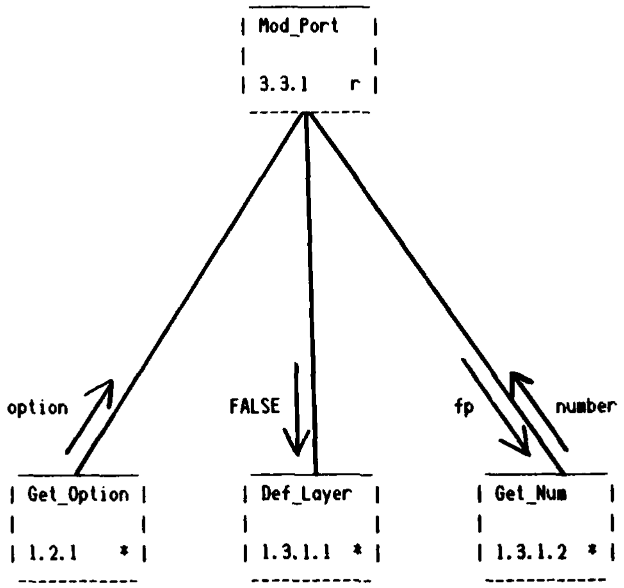
* This module defined under the given module number



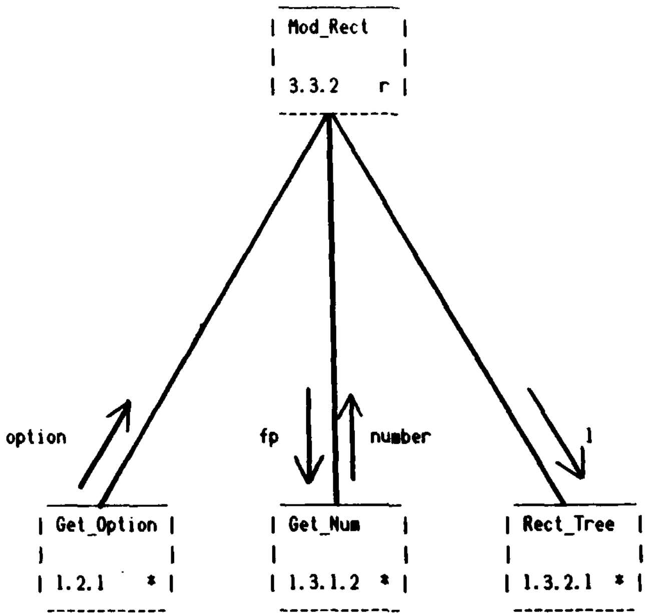
* This module defined under the given module number

r This module is recursive

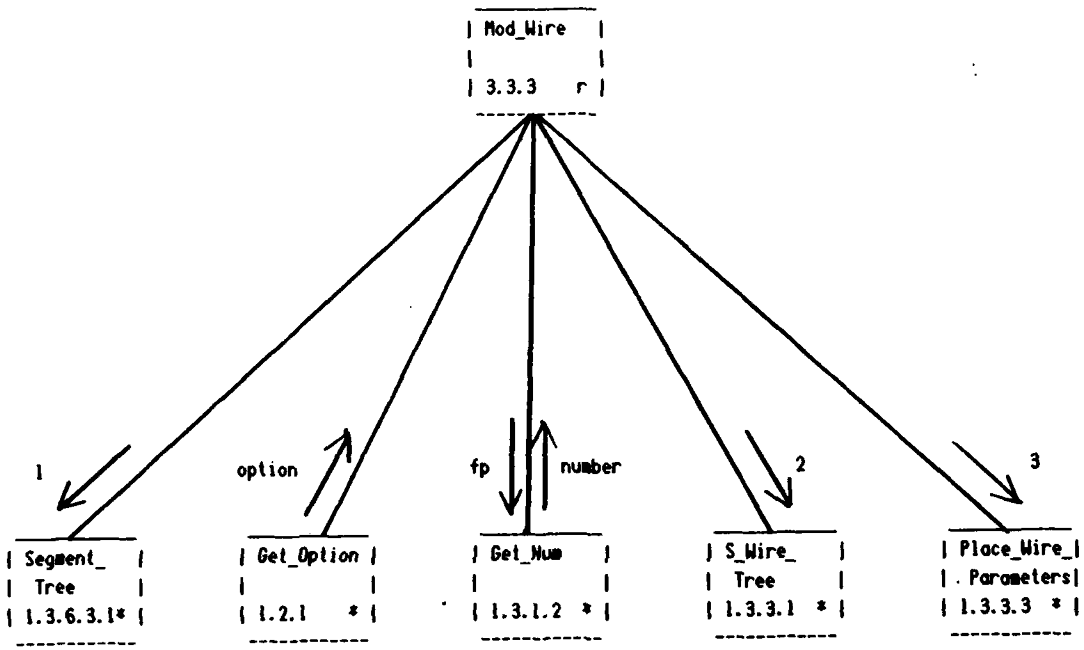
1. filename, cellname
2. exit, location, ptr to rectangles (f_rect, s_rect, p_rect, d_rect)
3. exit, location, wire_ptr
4. exit, location, ptr to vias (s_via, p_via, d_via)



* This module defined under the given module number
r This module is recursive



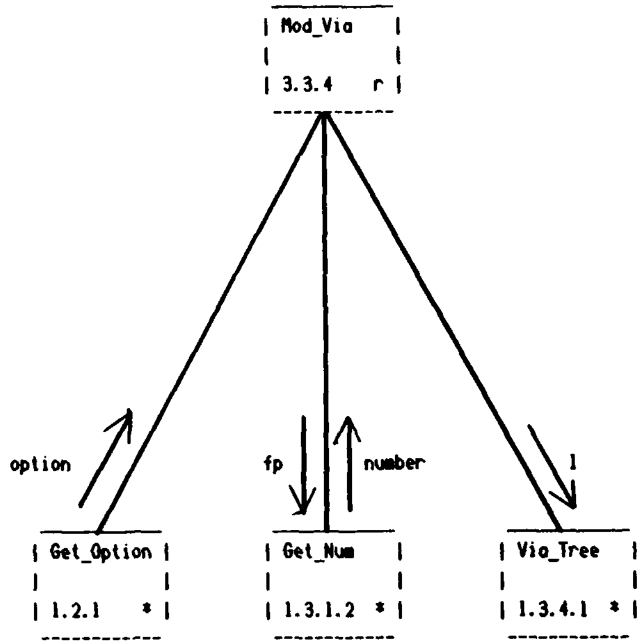
* This module defined under the given module number
r This module is recursive
1. r_h, r_l, r_x, r_y, ptr to rectangles
(f_rect, s_rect, p_rect, d_rect)



* This module defined under the given module number

r This module is recursive

1. p_width, p_layer, w_param
2. new_x, new_y, wire_ptr
3. width, length, direction, seg_ptr

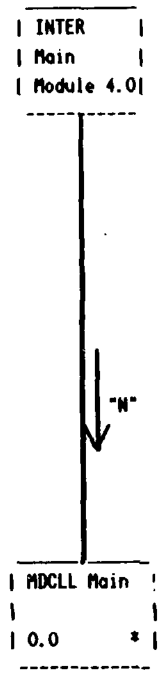


* This module defined under the given module number

r This module is recursive

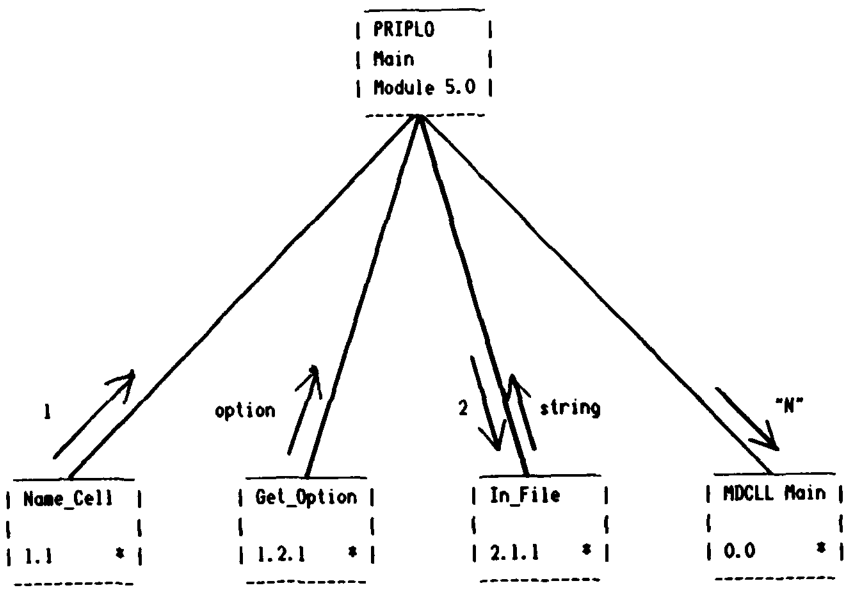
1. new_x, new_y, ptr to vias (s_via, p_via, d_via)

```
***************************************************************************
*                                                                         *
*       INTER Program Hierarchy:                                          *
*                                                                         *
*                                                                         *
*       4.0 INTER Main                                                    *
*         |                                                               *
*         |---- 0.0* MDCLL Main                                           *
*                                                                         *
*                                                                         *
***************************************************************************
```



* This module defined under the given module number

```
*************************************************************************
*                                                                       *
*       PRIPLO Program Hierarchy:                                       *
*                                                                       *
*                                                                       *
*       5.0  PRIPLO Main                                                *
*         |                                                             *
*         |---- 1.1* Name_Cell                                          *
*         |                                                             *
*         |---- 1.2.1* Get_Option                                       *
*         |                                                             *
*         |---- 2.1.1* In_File                                          *
*         |                                                             *
*         |---- 0.0* MDCLL Main                                         *
*                                                                       *
*************************************************************************
```

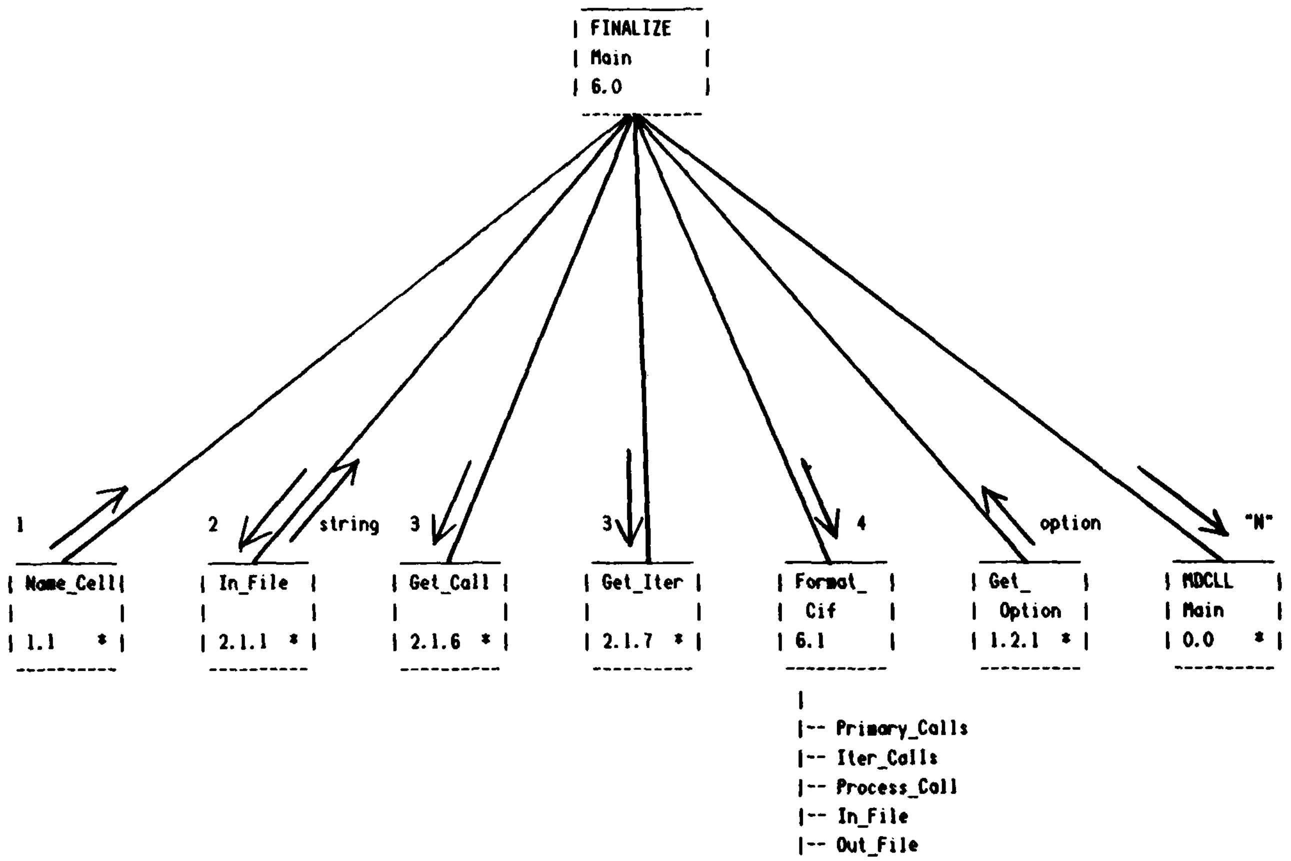


* This module defined under the given module number

1. on_disk, filename, cellname

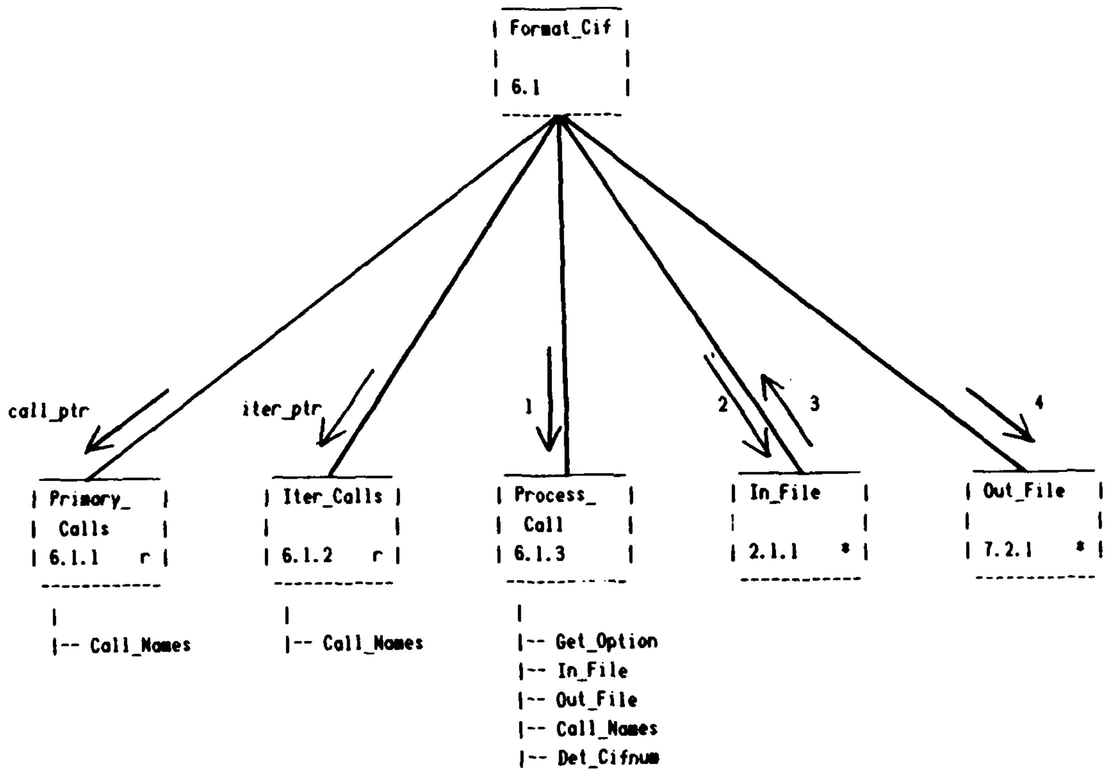
2. fp, character to be found, string

```
*************************************************************************
*                                                                       *
*       FINALIZE Program Hierarchy:                                     *
*                                                                       *
*                                                                       *
*       6.0  FINALIZE Main                                              *
*         |                                                             *
*         |---- 1.1* Name_Cell                                          *
*         |                                                             *
*         |---- 2.1.1* In_File                                          *
*         |                                                             *
*         |---- 2.1.6* Get_Call                                         *
*         |                                                             *
*         |---- 2.1.7* Get_Iter                                         *
*         |                                                             *
*         |---- 6.1  Format_Cif                                         *
*         |       |                                                     *
*         |       |---- 6.1.1r Primary_Calls                            *
*         |       |         |---- 6.1.1.1r Call_Names                   *
*         |       |                                                     *
*         |       |---- 6.1.2r Iter_Calls                               *
*         |       |         |---- 6.1.1.1* Call_Names                   *
*         |       |                                                     *
*         |       |---- 6.1.3  Process_Call                             *
*         |       |         |---- 1.2.1* Get_Option                     *
*         |       |         |---- 2.1.1* In_File                        *
*         |       |         |---- 7.2.1* Out_File                       *
*         |       |         |---- 6.1.1.1* Call_Names                   *
*         |       |         |---- 7.1* Det_Cifnum                       *
*         |       |                                                     *
*         |       |---- 2.1.1* In_File                                  *
*         |       |                                                     *
*         |       |---- 7.2.1* Out_File                                 *
*         |                                                             *
*         |---- 1.2.1* Get_Option                                       *
*         |                                                             *
*         |---- 0.0* MDCLL Main                                         *
*************************************************************************
```

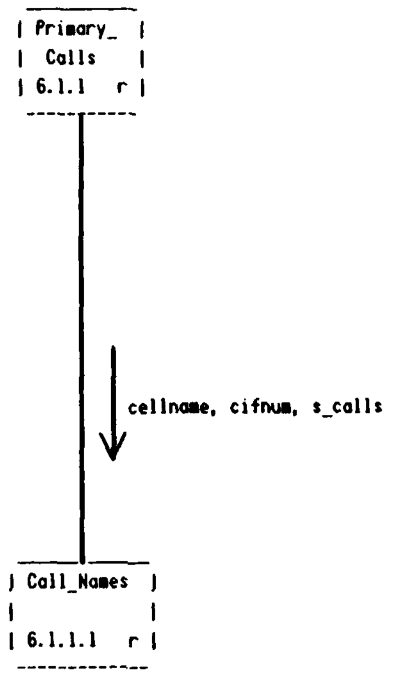


* This module defined under the given module number

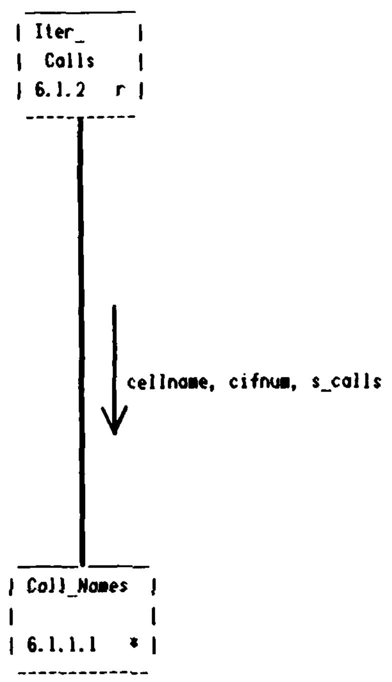
1. on_disk, filename, cellname
2. fp, string, character to be searched for
3. fp, string
4. filename, cellname



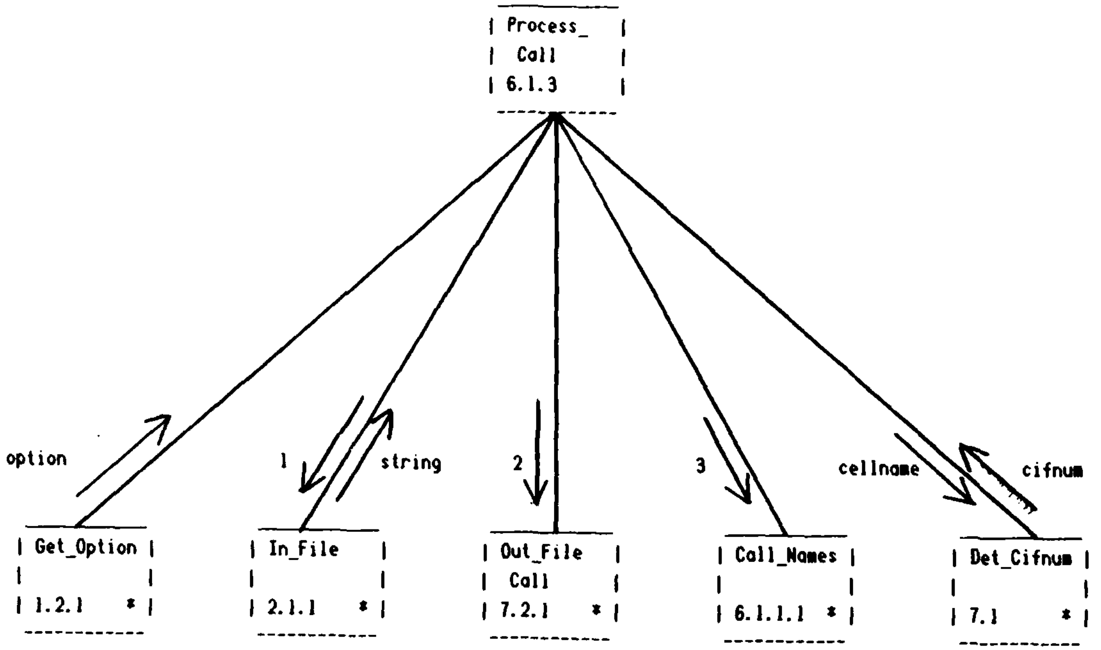
* This module defined under the given module number
r This module is recursive
1. cellname, fpl
2. fp2, string, character to be found
3. string
4. fpl, string, separator character



r This module is recursive



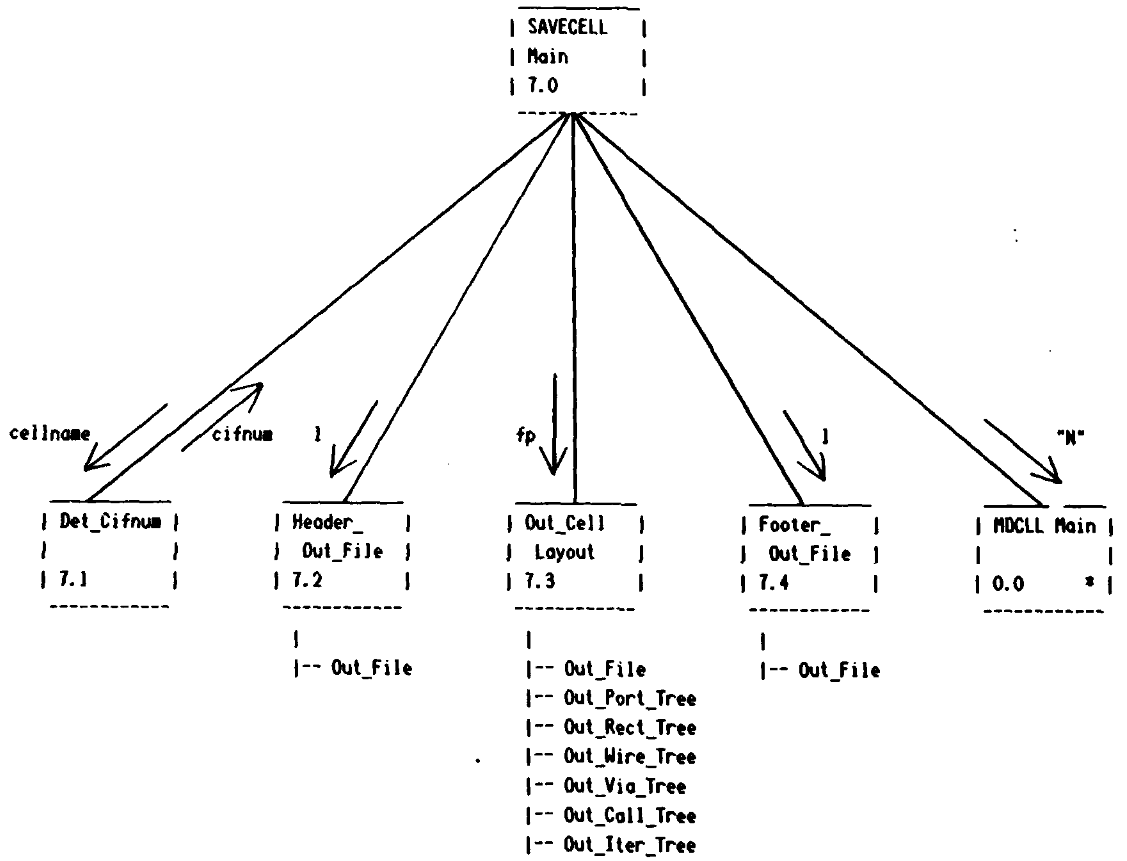
* This module defined under the given module number
r This module is recursive



* This module defined under the given module number

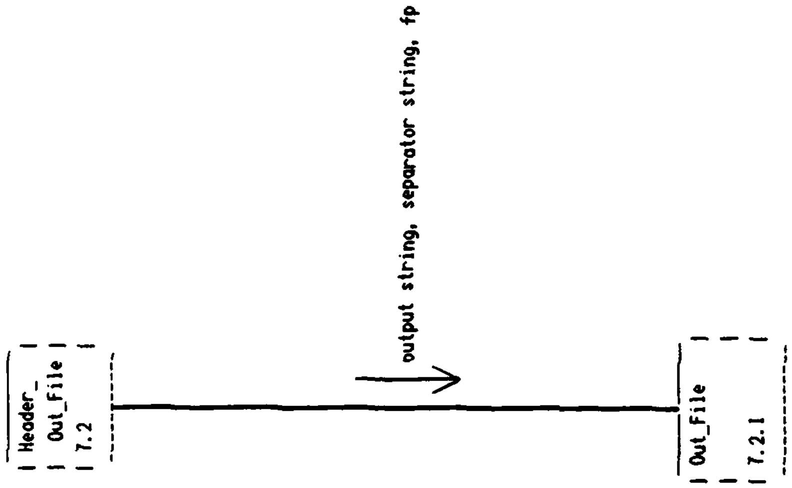
1. fp2, string, character to be searched for
2. fp1, string, separator character
3. cellname, cifnum, s_calls

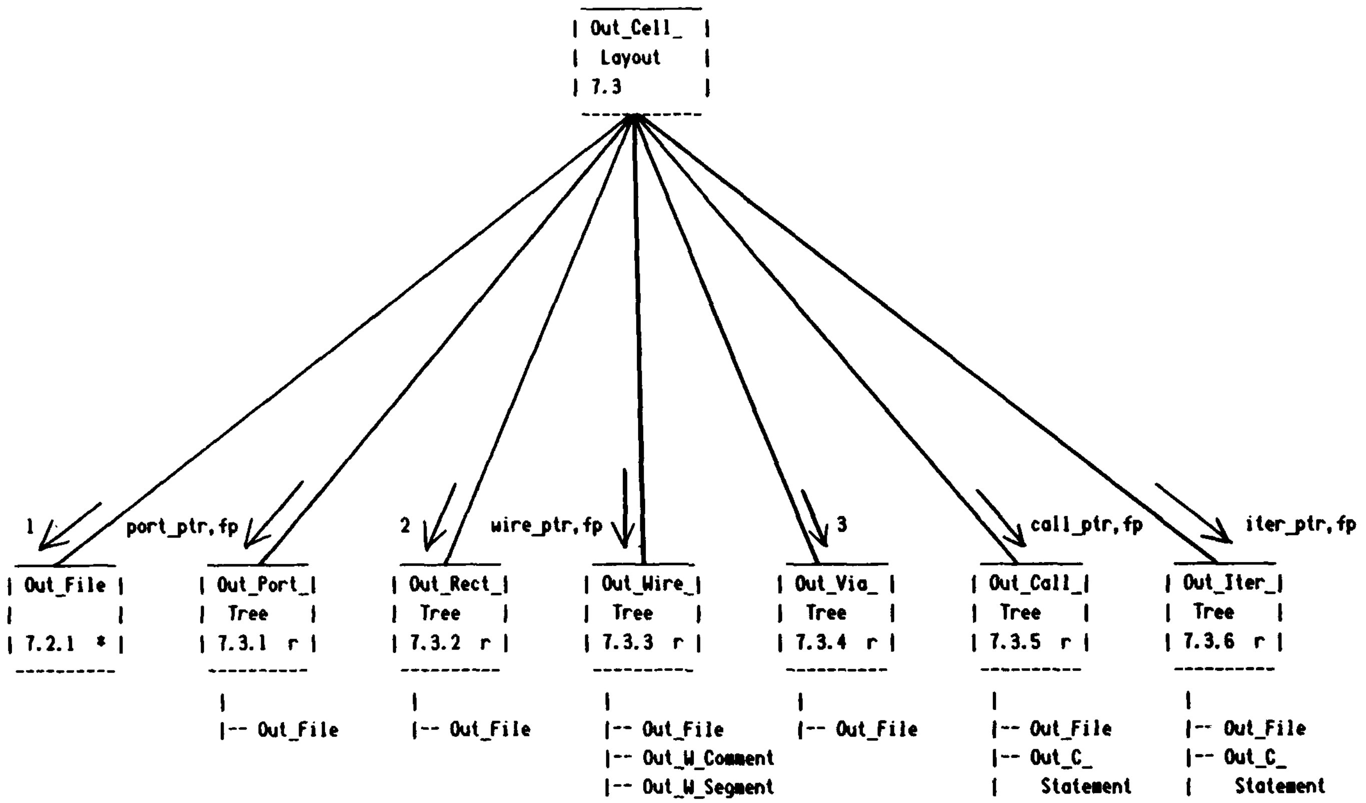
```
*************************************************************************
*                                                                       *
*       SAVECELL Program Hierarchy:                                     *
*                                                                       *
*                                                                       *
*       7.0  SAVECELL Main                                              *
*       |                                                               *
*       |---- 7.1  Det_Cifnum                                           *
*       |                                                               *
*       |---- 7.2  Header_Out_File                                      *
*       |       |---- 7.2.1 Out_File                                    *
*       |                                                               *
*       |---- 7.3  Out_Cell_Layout                                      *
*       |       |                                                       *
*       |       |---- 7.2.1* Out_File                                   *
*       |       |                                                       *
*       |       |---- 7.3.1r Out_Port_Tree                              *
*       |       |         |---- 7.2.1* Out_File                         *
*       |       |                                                       *
*       |       |---- 7.3.2r Out_Rect_Tree                              *
*       |       |         |---- 7.2.1* Out_File                         *
*       |       |                                                       *
*       |       |---- 7.3.3r Out_Wire_Tree                              *
*       |       |         |---- 7.2.1* Out_File                         *
*       |       |         |---- 7.3.3.1r Out_W_Comment                  *
*       |       |                     |---- 7.2.1* Out_File             *
*       |       |         |---- 7.3.3.2r Out_W_Segment                  *
*       |       |                     |---- 7.2.1* Out_File             *
*       |       |                     |---- 1.3.2.1* Rect_Tree          *
*       |       |                     |---- 1.3.4.1* Via_Tree           *
*       |       |                                                       *
*       |       |---- 7.3.4r Out_Via_Tree                               *
*       |       |         |---- 7.2.1* Out_File                         *
*       |       |                                                       *
*       |       |---- 7.3.5r Out_Call_Tree                              *
*       |       |         |---- 7.2.1* Out_File                         *
*       |       |         |---- 7.3.5.1 Out_C_Statement                 *
*       |       |         |           |---- 7.2.1* Out_File             *
*       |       |         |---- 7.3.5.2 Out_P_Names                     *
*       |       |                     |---- 7.2.1* Out_File             *
*       |       |                                                       *
*       |       |---- 7.3.6r Out_Iter_Tree                              *
*       |                 |---- 7.2.1* Out_File                         *
*       |                 |---- 7.3.5.1* Out_C_Statement                *
*       |                 |---- 7.3.5.2* Out_P_Names                    *
*       |                                                               *
*       |---- 7.4  Footer_Out_File                                      *
*       |       |---- 7.2.1* Out_File                                   *
*       |                                                               *
*       |---- 0.0* MDCLL Main                                           *
*                                                                       *
*************************************************************************
```



* This module defined under the given module number

1. cellname, a_cifnum, fp

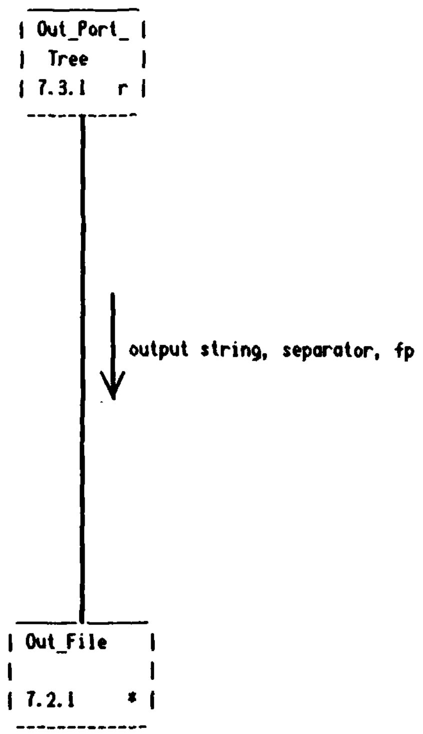

Header_ Out_File 7.2
output string, separator string, fp
Out_File 7.2.1



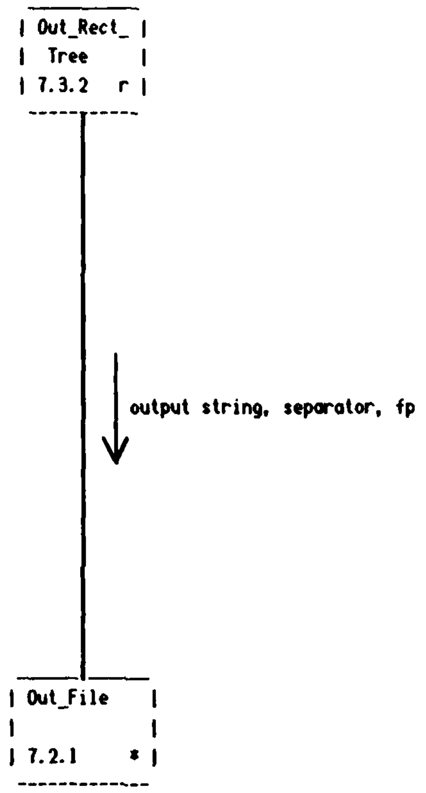
* This module defined under the given module number

r This module is recursive

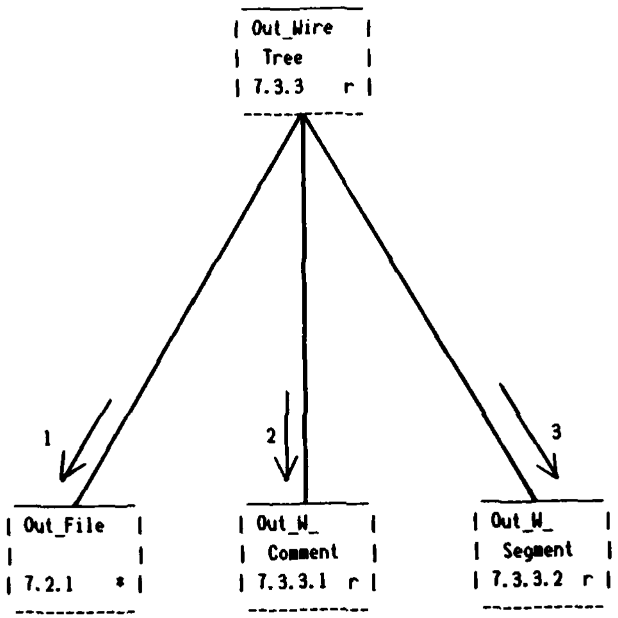
1. comment string, separator string, fp
2. fp, ptr to rectangles (f_rect, s_rect, d_rect, p_rect)
3. fp, via_size, ptr to vias (s_via, d_via, p_via)



* This module defined under the given module number
r This module is recursive

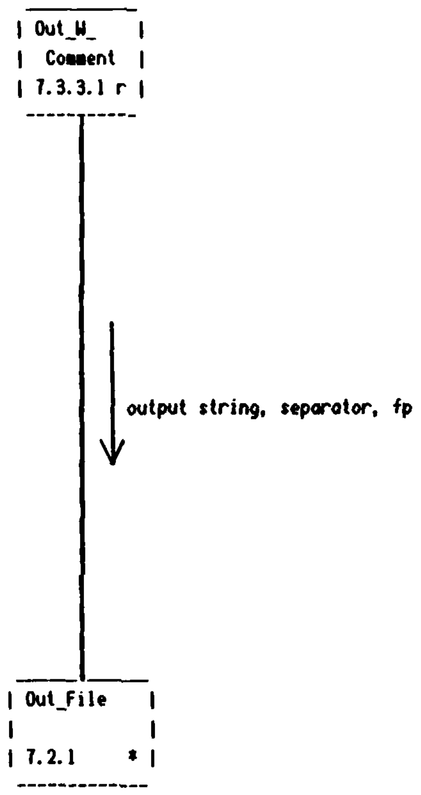


* This module defined under the given module number
r This module is recursive

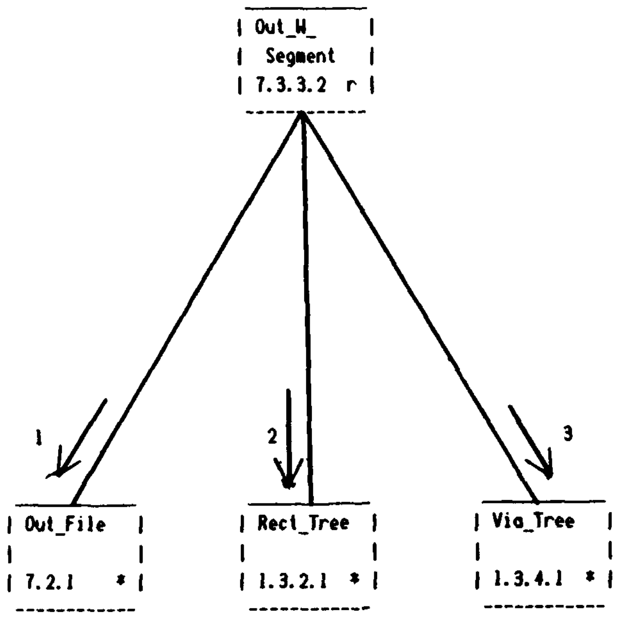


* This module defined under the given module number
r This module is recursive

1. output string, separator, fp
2. width, length, w_param, fp
3. w_x_location, w_y_location, w_param



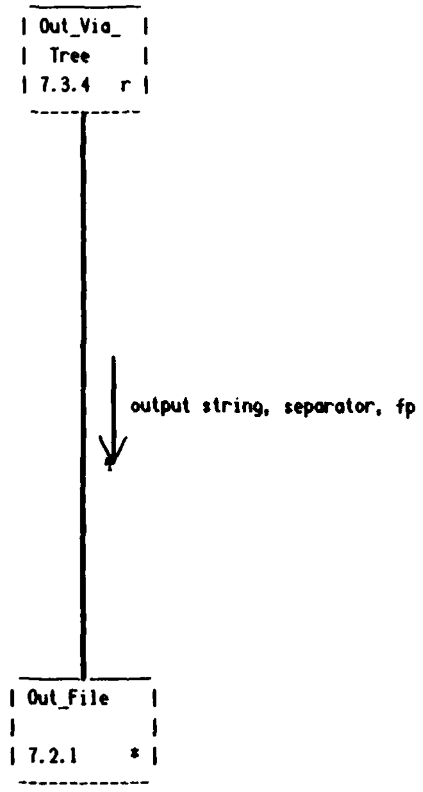
* This module defined under the given module number
r This module is recursive



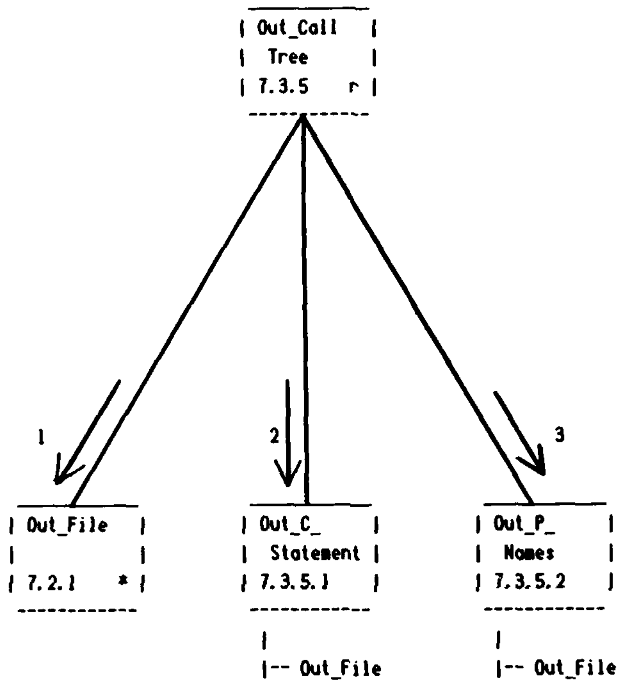
* This module defined under the given module number

r This module is recursive

1. ouput string
2. r_h, r_l, x_loc, y_loc, ptr to wire rectangles
   (w_f_rect, w_s_rect, w_p_rect, w_d_rect)
3. v_x, v_y, ptr to wire vias (w_s_via, w_p_via, w_d_via)



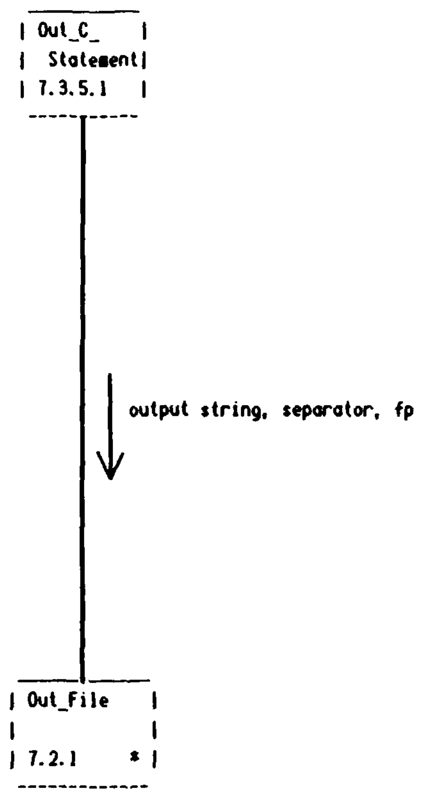
* This module defined under the given module number
r This module is recursive



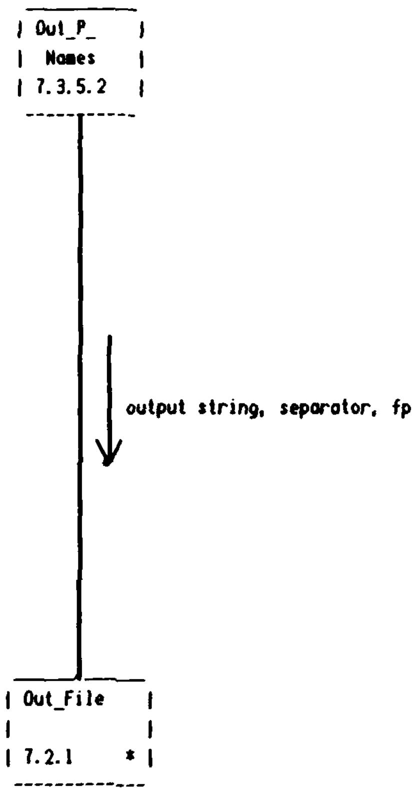
* This module defined under the given module number

r This module is recursive

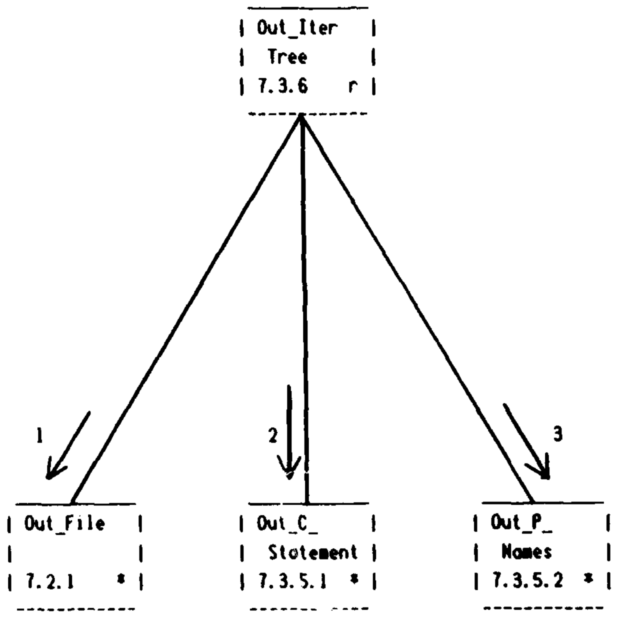
1. output string, separator, fp
2. cifnum, t_x, t_y, m_x, m_y, r_x, x_min, y_min, x_max, y_max, fp
3. c_ports, fp



* This module defined under the given module number



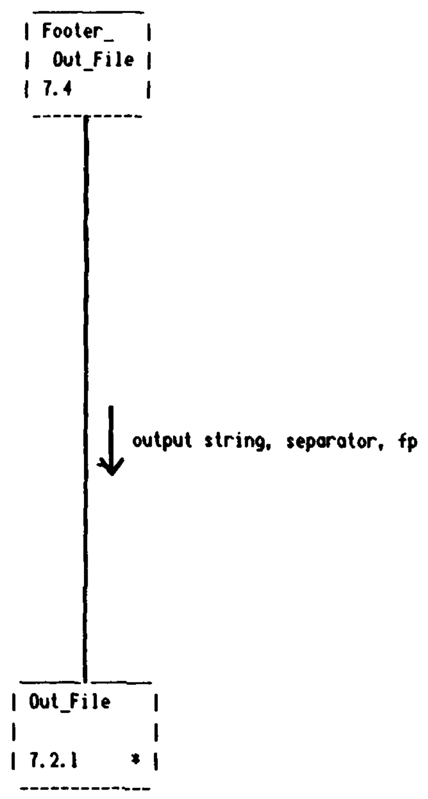
* This module defined under the given module number



* This module defined under the given module number

r This module is recursive

1. output string, separator, fp
2. cifnum, t_x, t_y, m_x, m_y, r_x, x_min, y_min, x_max, y_max, fp
3. c_ports, fp



* This module defined under the given module number

Appendix B

MDCLL Program Listing

This appendix contains a listing of the standard header used throughout the MDCLL system followed by each of the programs.

```
/*****************************************************************/
/*                                                               */
/*         STANDARD HEADER FOR USE IN THE MDCLL SYSTEM           */
/*                                                               */
/*****************************************************************/


/*****************************************************************/
/*                                                               */
/*  The file tprintf.c is included to allow output to the user   */
/*  by way of the C language statement "printf"                  */
/*                                                               */
#include "tprintf.c"                                           /**/
/*                                                               */
/*****************************************************************/


/*****************************************************************/
/*                                                               */
/*  The file exec.c is included to permit the execution of       */
/*  one program from the interior of another                     */
/*                                                               */
#include "exec.c"                                              /**/
/*                                                               */
/*****************************************************************/


/*****************************************************************/
/*                                                               */
/*  Following are the global constants used in the MDCLL system */
/*  The first five are here since they are used in many of the   */
/*    programs in the system; the rest are in the header to     */
/*    highlight them since they will most likely need changing. */
/*                                                               */
#define FILE int                                               /**/
#define EOF -1                                                 /**/
#define TRUE 1                                                 /**/
#define FALSE 0                                                /**/
#define NULL 0                                                 /**/
#define SCALE_A "50"                                           /**/
#define SCALE_B "1"                                            /**/
#define M_VIA_SIZE 8                                           /**/
#define M2_VIA_SIZE 8                                          /**/
#define D_VIA_SIZE 8                                           /**/
#define C_VIA_SIZE 4                                           /**/
#define V_VIA_SIZE 4                                           /**/
#define P_VIA_SIZE 10                                          /**/
#define AVAIL_SPACE 20480                                      /**/
/*                                                               */
/*****************************************************************/
```

```
/*****************************************************************/
/*                                                               */
/*  Following are the global variables and initial values       */
/*                                                               */
char    layer = 'P';                                          /**/
char    *seg_ptr = NULL;                                      /**/
char    *c_port_ptr = NULL;                                   /**/
char    *i_call_ptr = NULL;                                   /**/
int     min_x_b = 9999;                                       /**/
int     min_y_b = 9999;                                       /**/
int     max_x_b = -9999;                                      /**/
int     max_y_b = -9999;                                      /**/
int     space_used = 0;                                       /**/
/*                                                               */
/*****************************************************************/
```

```
/*****************************************************************/
/*                                                               */
/*  Following are the Structures for storing CIF in memory      */
/*                                                               */
/*****************************************************************/
/**/
/**/    struct inter_ports
/**/            {
/**/            char    port_name [15],
/**/                    port_layer;
/**/            int     x_loc, y_loc;
/**/            struct  inter_ports *next_port;
/**/            }
/**/            *port_ptr = NULL;
/**/
/**/    struct rectangle
/**/            {
/**/            int     height,
/**/                    length,
/**/                    x_location,
/**/                    y_location;
/**/            struct  rectangle
/**/                    *less_than_xl
/**/                    *equals_x,
/**/                    *greater_than_x;
/**/            }
/**/            *f_rect, *s_rect, *d_rect, *p_rect,
/**/            *w_f_rect, *w_s_rect, *w_d_rect, *w_p_rect = NULL;
/**/
```

```
/**/    struct via
/**/            {
/**/            int     v_x_location,
/**/                    v_y_location;
/**/            struct via
/**/                    *v_l_t_x,
/**/                    *v_e_x,
/**/                    *v_g_t_x;
/**/            }
/**/            *s_via, *d_via, *p_via,
/**/            *w_s_via, *w_d_via, *w_p_via = NULL;
/**/
/**/    struct wire
/**/            {
/**/            int     w_x_location,
/**/                    w_y_location;
/**/            struct wire_parameters
/**/                    {
/**/                    int     width,
/**/                            length;
/**/                    char    w_layer,
/**/                            direction;
/**/                    struct wire_parameters
/**/                            *next_segment;
/**/                    }
/**/                    *w_param;
/**/            struct wire
/**/                    *w_l_t_x,
/**/                    *w_e_x,
/**/                    *w_g_t_x;
/**/            }
/**/            *wire_ptr = NULL;
/**/
/**/    struct call_statement
/**/            {
/**/            char    cellname[15];
/**/            int     cifnum,
/**/                    tran_x, tran_y,
/**/                    mir_x, mir_y,
/**/                    rot_x,
/**/                    x_min, y_min, x_max, y_max;
/**/            struct named_ports
/**/                    {
/**/                    char    port_name [15];
/**/                    struct named_ports
/**/                            *next_name;
/**/                    }
/**/                    *c_ports;
/**/            struct call_statement
/**/                    *l_t_name,
/**/                    *e_name,
/**/                    *g_t_name;
/**/            }
/**/            *call_ptr = NULL;
/**/
/**/    struct iter_statement
/**/            {
/**/            char    cellname [15];
/**/            int     cifnum,
/**/                    x_iters, y_iters,
/**/                    x_pitch, y_pitch;
/**/            struct call_statement
/**/                    *i_call;
/**/            struct iter_statement
/**/                    *l_t_cifnum, *e_cifnum, *g_t_cifnum;
/**/            }
/**/            *iter_ptr = NULL;
/**/
/**/    struct called_cells
/**/            {
/**/            char    cellname [15];
/**/            int     cifnum;
/**/            struct  called_cells
/**/                    *next_call;
/**/            }
/**/            *s_calls = NULL;
/**/
/*******************  End of Structure for storing CIF ************/
```

```
/***********************************************************************
*                                                                      *
*       Program Name:  MDCLL.C                                         *
*                                                                      *
*       Version/Date:  1.0  20 July 1985                               *
*                                                                      *
*       Function:  This program is a Menu Driven Chip Layout           *
*               Language System for use in the design of VLSI          *
*               circuits.  Its final product is a file in the          *
*               Caltech Intermediate Form.                             *
*                                                                      *
*       Usage:  This program must be used with its subprograms:        *
*               CREATE, DPM, MODIFY, INTER, PRIPLO, and FINAL.         *
*               Each program is coded in C/80 and must be compiled     *
*               and assembled accordingly.                             *
*                                                                      *
*       Include Files Needed:  STDLIB.C (including toupper, tolower    *
*               alloc, sizeof, atoi, itoa), MDCLL.H (TPRINTF.C, EXEC.C)*
*                                                                      *
*       Author:  Steven C. Morrese, Capt, USAF                         *
*                                                                      *
*       Program Hierarchy:                                             *
*                                                                      *
*       0.0  MDCLL Main                                                *
*         |                                                            *
*         |---- 0.1  Initialize                                        *
*         |                                                            *
*         |---- 0.2  Main_Menu                                         *
*         |                                                            *
*         |---- 0.3  Fetch_Run                                         *
*                 |                                                    *
*                 |----   1.0* CREATE Main                             *
*                 |                                                    *
*                 |----   2.0* DPM Main                                *
*                 |                                                    *
*                 |----   3.0* MODIFY Main                             *
*                 |                                                    *
*                 |----   4.0* INTER Main                              *
*                 |                                                    *
*                 |----   5.0* PRIPLO Main                             *
*                 |                                                    *
*                 |----   6.0* FINALIZE Main                           *
*                                                                      *
***********************************************************************/

#include "mdcll.h"       /*    Standard header used in every program  */
                         /* in the MDCLL system even though not every */
                         /* program uses every item defined.          */
```

```
/************************************************************************
*                                                                       *
*       Name:  Main                                                     *
*       Module Number: 0.0                                              *
*       Version/Date: 1.0   20 July 1985                                *
*       Input Parameters:  Initialize variable                          *
*       Output Parameters:  none                                        *
*       Modules Called:  Initialize, Main_Menu, Fetch_Run               *
*       Calling Modules:  SAVECELL Main, FINALIZE Main                  *
*       Function:                                                       *
*                 Control the selection and running of the other        *
*                 programs in the MDCLL system.                         *
*       PDL:                                                            *
*         If initialization message wanted                              *
*           then Initialize;                                            *
*         Display Main Menu and get response;                           *
*         If not quit command                                           *
*           then execute requested subprogram.                          *
*                                                                       *
************************************************************************/

main (argc,argv)

int argc;
char **argv[];
{
char option;

option = ' ';
if (argv[1][0] != 'N')
  Initialize ();
option = Main_Menu ();
if (option != 'Q')
  Fetch_Run (option);

}
```

```
/************************************************************************
*                                                                        *
*       Module Name:  Initialize                                         *
*       Module Number:  0.1                                              *
*       Version/Date:  1.0  20 July 1985                                 *
*       Input parameters: none                                           *
*       Output parameters: none                                          *
*       Globals Used: none                                               *
*       Modules Called:  none                                            *
*       Calling Modules:  Main(0.0)                                      *
*       Function:  Print introductory message.                           *
*       PDL:  N/A                                                        *
*                                                                        *
************************************************************************/

Initialize ()

{
printf ("\n\n\n\n\n\n\n\n\n\n\n\n\n");
printf ("\n Welcome to the world of Doctor Chill \n\n");
printf ("\n   If you don't want to get these instructions ");
printf ("\n simply type an N after the program name; 'MDCLL N'. \n");
printf ("\n Anytime a question is asked and return or N is ");
printf ("\n requested any character will suvvice for the N.\n");


printf ("\n \n ADD instructions on using the system here \n\n");


printf ("\n\n \n\n Type return to begin: ");
while (getc(0) != '\n');
printf ("\n\n\n\n\n\n\n\n\n\n\n\n");
}
```

```
/***********************************************************************
*                                                                      *
*       Module Name:  Main_Menu                                        *
*       Module Number:  0.2                                            *
*       Version/Date:  1.0  20 July 1985                               *
*       Input Parameters:  none                                        *
*       Output Parameters:  option: subprogram to be run               *
*       Globals Used: none                                             *
*       Modules Called:  none                                          *
*       Calling Modules:  Main (0.0)                                   *
*       Function:  Display the available options and ensure a valid    *
*                response is obtained from the user                    *
*       PDL:                                                           *
*         While response not a valid option                            *
*           Display Menu,                                              *
*           get response,                                              *
*         return option.                                               *
*                                                                      *
***********************************************************************/
Main_Menu ()
{
char    option;
int     option_valid;
option_valid = FALSE;
while (option_valid == FALSE)
  {
  printf ("\n \t\t MDCLL Main Menu \n\n");
  printf ("\n Options available are \n");
  printf ("\n\t C = Create a new cell \n");
  printf ("\n\t D = Delete, place, or move an existing cell \n");
  printf ("\n\t M = Modify an existing cell \n");
  printf ("\n\t I = Interconnect cells already placed \n");
  printf ("\n\t P = Print or Plot \n");
  printf ("\n\t F = Finalize circuit \n");
  printf ("\n\t Q = Quit to the operating system \n");
  printf ("\n\n Select Option (C,D,M,I,P,F,Q): ");
  option = getc(0);
  if (option != '\n')
    while (getc(0) != '\n');
  printf ("\n\n\n\n\n\n\n\n\n");
  option = toupper (option);
  switch (option)
    {
    case 'C' :    case 'D' :
    case 'M' :    case 'I' :
    case 'P' :    case 'F' :
    case 'Q' : option_valid = TRUE; break;
    default  : printf ("\n Option not valid = %c\n\n", option);
    }
  }
return (option);
}
```

```
/**********************************************************************
*                                                                     *
*      Module Name:  Fetch_Run                                        *
*      Module Number:  0.3                                            *
*      Version/Date:  1.0  20 July 1985                               *
*      Input Parameters:  option:  subprogram to be run               *
*      Output Parameters:  none                                       *
*      Globals Used:  none                                            *
*      Modules Called:  Subprograms in MDCLL system                   *
*      Calling Modules:  Main (0.0)                                   *
*      Function:  Ensure the requested subprogram is available        *
*             and then execute it.                                    *
*      PDL:                                                           *
*        Set filename to be executed;                                 *
*          try to access file from on disk,                           *
*          while file not available and filename valid                *
*            print error message and get response,                    *
*            if user wants to quit                                    *
*              then set file validity to false                        *
*              else check if file available on disk;                  *
*          if file still valid                                        *
*            then execute the subprogram.                             *
*                                                                     *
**********************************************************************/
Fetch_Run (option)
{
char    *filename;
int     file_valid;
FILE    *fopen(),*fp;
file_valid = TRUE;
if (option == 'C')  filename = "CREATE.COM";
else if (option == 'D')  filename = "DPM.COM";
else if (option == 'M')  filename = "MODIFY.COM";
else if (option == 'I')  filename = "INTER.COM";
else if (option == 'P')  filename = "PRIPLO.COM";
else if (option == 'F')  filename = "FINALIZE.COM";
fp = fopen (filename, "r");
while ((fp == NULL) && (file_valid == TRUE))
  {
  printf ("Place disk with file %s in the active drive\n\n", filename);
  printf ("Hit RETURN to continue or Q to exit system: ");
  if (getc(0) != 'Q')
    file_valid = FALSE;
    else fp = fopen (filename, "r");
  }
if (file_valid = TRUE)
  {
  fclose (fp);
  exec (filename, " ");
  }
}
#include "stdlib.c"
```

```
/***********************************************************************
*                                                                      *
*       Program Name:  CREATE.C                                        *
*       Version/Date:  1.0  22 July 1985                               *
*                                                                      *
*       Function:  This subprogram to MDCLL.C creates a new cell in    *
*               the Caltech Intermediate form.                         *
*                                                                      *
*       Library Files Needed:  STDLIB.C, PRINTF.C, SCANF.C, EXEC.C     *
*                                                                      *
*       Program Hierarchy:  See next page                              *
*                                                                      *
***********************************************************************/

#include "mdcll.h"



/****************************************************************
    the following filler was added to ensure data is not
        overwritten when the subprogram SAVECELL is executed.
****************************************************************/

#asm
filler: DS      2500H
#endasm
```

```
/**********************************************************************
*       1.0  CREATE Main                                              *
*         |                                                           *
*         |---- 1.1  Name_cell                                        *
*         |                                                           *
*         |---- 1.2  C_Menu                                           *
*         |         |---- 1.2.1  Get_Option                           *
*         |                                                           *
*         |---- 1.3  Implement                                        *
*                   |                                                 *
*                   |---- 1.3.1  Def_Port                             *
*                   |          |---- 1.3.1.1  Def_Layer               *
*                   |          |---- 1.3.1.2  Get_Num                 *
*                   |          |---- 1.3.1.3r Port_Tree               *
*                   |          |---- 1.2.1* Get_Option                *
*                   |                                                 *
*                   |---- 1.3.2  Def_Rectangle                        *
*                   |          |---- 1.3.1.1* Def_Layer               *
*                   |          |---- 1.3.1.2* Get_Num                 *
*                   |          |---- 1.3.2.1r Rect_Tree               *
*                   |          |---- 1.2.1* Get_Option                *
*                   |                                                 *
*                   |---- 1.3.3  Def_Wire                             *
*                   |          |---- 1.3.1.1* Def_Layer               *
*                   |          |---- 1.3.1.2* Get_Num                 *
*                   |          |---- 1.3.3.1r S_Wire_Tree             *
*                   |          |---- 1.3.3.2  Get_W_Option            *
*                   |          |---- 1.3.3.3r Place_Wire_Parameters   *
*                   |          |---- 1.2.1* Get_Option                *
*                   |                                                 *
*                   |---- 1.3.4  Def_Via                              *
*                   |          |---- 1.3.1.1* Def_Layer               *
*                   |          |---- 1.3.1.2* Get_Num                 *
*                   |          |---- 1.3.4.1r Via_Tree                *
*                   |          |---- 1.2.1* Get_Option                *
*                   |                                                 *
*                   |---- 1.3.5  Def_Flash                            *
*                   |                                                 *
*                   |---- 1.3.6  Print_Work                           *
*                   |          |---- 1.2.1* Get_Option                *
*                   |          |---- 1.3.6.1r Walk_P_Tree             *
*                   |          |---- 1.3.6.2r Walk_R_Tree             *
*                   |          |---- 1.3.6.3r Walk_W_Tree             *
*                   |          |            |---- 1.3.6.3.1r Segment_Tree  *
*                   |          |---- 1.3.6.4r Walk_Via_Tree           *
*                   |          |---- 1.3.6.5r Walk_C_Tree             *
*                   |          |---- 1.3.6.6r Walk_I_Tree             *
*                   |                                                 *
*                   |---- 1.3.7  Save_Cell                            *
*                              |---- 1.3.7.1  Store_Ptrs              *
*                              |---- 7.0  SAVECELL Main               *
**********************************************************************/
```

```
/************************************************************************
*                                                                       *
*       Module Name: Main                                               *
*       Module Number: 1.0                                              *
*       Version/Date: 1.0  21 Aug 1985                                  *
*       Input Parameters: none                                          *
*       Output Parameters: none                                         *
*       Globals Used:  none                                             *
*       Moaules Called: Name_Cell, C_Menu, Implement                    *
*       Calling Modules: MDCLL program                                  *
*       Function:                                                       *
*               Name a new cell, Display options available, and then    *
*               implement the desired option (place the needed wires    *
*               in the physical layout of a VLSI circuit).              *
*       PDL:                                                            *
*         While name is not already in use                              *
*           Determine name for new cell;                                *
*         Display menu and get option;                                  *
*         While option other than quit                                  *
*           Implement desired option,                                   *
*           Display menu and get option.                                *
*                                                                       *
************************************************************************/

main()
{
int     name_valid, on_disk;
char    option,
        cellname [15],  filename [12];

name_valid = FALSE;
while (name_valid == FALSE)
  {
  printf ("\n\n Enter cellname for new cell: ");
  Name_Cell (&on_disk, filename, cellname);
  if (on_disk == FALSE)
    {
    printf ("\nCell %s will be stored as file %s\n", cellname, filename);
    name_valid = TRUE;
    }
    else printf ("\n\n Name %s already in use.\n\n", cellname);
  }
while ((option = C_Menu (cellname)) != 'Q')
    Implement (option, filename, cellname);
}

#include "cre8lib.lib"
#include "stdlib.c"
```

```
#ifneed Name_Cell
/************************************************************************
*                                                                       *
*       Module Name:  Name_Cell                                         *
*       Module Number:  1.1                                             *
*       Version/Date:  1.0   21 Aug 1985                                *
*       Input Parameters: none                                          *
*       Output Parameters: on_disk, cellname, filename                  *
*       Globals Used:  none                                             *
*       Modules Called:  none                                           *
*       Calling Modules: CREATE Main, D_Implement, MODIFY Main,         *
*                  PRIPLO Main, FINALIZE Main                           *
*       Function:  Name a new cell, ensure the name isn't already       *
*               in use, and initialize the output file on disk.         *
*       PDL:                                                            *
*           Get cellname from user;                                     *
*           Obtain filename from cellname for new cell;                 *
*           If filename already in use                                  *
*             then set on_disk = TRUE,                                  *
*             else filename not yet used (on_disk = FALSE).             *
************************************************************************/
Name_Cell (on_disk, filename, cellname)
int     *on_disk;
char    *filename,
        *cellname;
{
int     i;
FILE    *fopen(),*fp;
i = 0;
while ((cellname[i] = toupper (getc(0))) != '\n')
  ++i;
cellname[i] = '\0';
i = 0;
while ((filename[i] = cellname[i]) != '\0')
  ++i;
filename[i] = '.'; filename[++i] = 'C';
filename[++i] = 'I'; filename[++i] = 'F';
filename[++i] = '\0';
if (cellname[1] == ':')
  {
  i = 0;
  while ((cellname[i] = cellname[2 + i]) != '\0')  ++i;
  }
fp = fopen (filename, "r");
if (fp != NULL)
  {
  fclose (fp);
  *on_disk = TRUE;
  }
  else *on_disk = FALSE;
}
#endif
```

```
#ifneed C_Menu
/************************************************************************
*                                                                      *
*       Module Name: C_Menu                                            *
*       Module Number: 1.2                                             *
*       Version/Date: 1.0  22 July 1985                                *
*       Input Parameters: cellname                                     *
*       Output Parameters: option                                      *
*       Globals Used: space_used, AVAIL_SPACE                          *
*       Modules Called: Get_Option                                     *
*       Calling Modules: Main (1.0)                                    *
*       Function:  Obtain a valid option from the user                 *
*       PDL:                                                           *
*         While not a valid option                                     *
*           Display options;                                           *
*           Get option;                                                *
*           check validity of option;                                  *
*         return option.                                               *
*                                                                      *
************************************************************************/
C_Menu (cellname)
char    *cellname;
{
char    option;
int     option_valid,
        space_left;

option_valid = FALSE;
while (option_valid == FALSE)
  {
  space_left = AVAIL_SPACE - space_used;
  printf ("\n\n\t\t\tMemory used: %d  space left: %d\n",space_used,space_left);
  printf ("\nOptions available for cell %s are:\n\n", cellname);
  printf ("\nDefine a :\t\t\tor:\n");
  printf ("\n  R = Rectangle (box)\t\t  P = Print work accomplished\n");
  printf ("\n  W = Wire           \t\t  E = Exit to main menu\n");
  printf ("\n  V = Via            \t\t      after saving file\n");
  printf ("\n  F = round Flash    \t\t  Q = Quit to system\n");
  printf ("\n  I = Interconnection Port\n");
  printf ("\n\t\t\tCurrent default layer is %c\n", layer);
  printf ("\nSelect Option (R,W,V,F,I,P,E,Q): ");
  option = Get_Option ();
  if (option == 'R' || option == 'P' || option == 'W'
    ||option == 'V' || option == 'F' || option == 'I'
    ||option == 'Q' || option == 'E')
    option_valid = TRUE;
    else  printf ("Option not valid\n\n");
  }
printf ("\n\n\n\n\n\n\n\n\n");
return (option);
}
#endif
```

```
#ifneed Get_Option
/************************************************************************
*                                                                       *
*       Module Name:  Get_Option                                        *
*       Module Number:  1.2.1                                           *
*       Version/Date:  1.0   5 August 1985                              *
*       Input Parameters:  none                                         *
*       Output Parameters:  option                                      *
*       Globals Used: none                                              *
*       Modules Called:  none                                           *
*       Calling Modules:  C_Menu, Def_Layer, Def_Rectangle, Def_Wire,   *
*         Def_Via, Print_Work, Print_Rect, Print_Wire,  Def_Port,       *
*         Print_Via, DPM_Main, D_Implement, Call_Parameters, Place_Call *
*         C_Port_Names, Mod_Call, Mod_Iter, MODIFY Main, Det_Mod,       *
*         M_Menu, Mod_Port, Mod_Rect, Mod_Wire, Mod_Via, PRIPLO Main,   *
*         FINALIZE Main, Process_Call                                   *
*       Function:  Get a single character option from the terminal      *
*       PDL:                                                            *
*         Get a character from the user;                                *
*         Convert it to upper case;                                     *
*         Continue geeting characters until a carriage return;          *
*         return the first character.                                   *
*                                                                       *
************************************************************************/

Get_Option ()

{
char    option;

option = toupper (getc(0));
if (option != '\n')
  while (getc(0) != '\n');
return (option);
}

#endif
```

```
#ifneed Implement
/*************************************************************************
*                                                                        *
*       Module Name:  Implement                                          *
*       Module Number:  1.3                                              *
*       Version/Date:  1.0  23 July 1985                                 *
*       Input Parameters:  option, filename, cellname                    *
*       Output Parameters:  none                                         *
*       Modules Called:  Def_Port, Def_Rectangle, Def_Wire, Def_Via,     *
*                  Def_Flash, Print_Work, Save_Cell                      *
*       Calling Modules:  Main (1.0)                                     *
*       Function:  Process requested option                              *
*       PDL:  N/A                                                        *
*                                                                        *
*************************************************************************/

Implement (option, filename, cellname)
char    option,
        *filename,
        *cellname;
{
if (option == 'I') Def_Port ();
else if (option == 'R') Def_Rectangle ();
else if (option == 'W') Def_Wire ();
else if (option == 'V') Def_Via ();
else if (option == 'F') Def_Flash ();
else if (option == 'P') Print_Work ();
else if (option == 'E') Save_Cell (filename, cellname);
}

#endif
```

```
#ifneed Def_Port
/************************************************************************
*       Module Name:  Def_Port                                          *
*       Module Number:  1.3.1                                           *
*       Version/Date:  1.0  7 September 1985                            *
*       Input Parameters:  none                                         *
*       Output Parameters:  none                                        *
*       Globals Used: port_ptr                                          *
*       Modules Called: Def_Layer, Get_Num, Port_Tree, Get_Option       *
*       Calling Modules: Implement                                      *
*       Function:  Define an interconnection port                       *
*       PDL:                                                            *
*         While not finished placing ports                              *
*           Enter port name, location, and layer,                       *
*           Place port in structure.                                    *
*                                                                       *
************************************************************************/


Def_Port ()
{
char    portname [15];
int     i, finished, p_x, p_y;

printf ("\n\n\n\n\t\t Ready to place an Interconnection Port \n\n\n\n");
finished = FALSE;
while (finished == FALSE)
  {
  printf ("\n Enter the port's name: ");
  i = 0;
  while ((portname [i] = getc (0)) != '\n') ++i;
  portname [i] = '\0';
  printf ("\n Enter the port layer \n");
  Def_Layer (FALSE);
  printf ("\n\t\t X Location: ");
  p_x = Get_Num ();
  printf ("\n\t\t Y Location: ");
  p_y = Get_Num ();
  Port_Tree (portname, p_x, p_y, &port_ptr);
  printf ("\n Do you want to place another port (ret or N): ");
  if (Get_Option () != '\n')  finished = TRUE;
  }
}

#endif
```

```
#ifneed Def_Layer
/************************************************************************
*                                                                       *
*       Module Name:  Def_Layer                                         *
*       Module Number:  1.3.1.1                                         *
*       Version/Date:  1.0  23 July 1985                                *
*       Input Parameters:  layer_valid                                  *
*       Output Parameters: none                                         *
*       Globals Used:  layer                                            *
*       Modules Called:  Get_Option                                     *
*       Calling Modules:  Def_Rectangle, Def_Port, Def_Wire, Def_Via,   *
*                  M_Implement, Mod_Port                                *
*       Function:  Change the default layer                             *
*       PDL:                                                            *
*         if already have a valid layer                                 *
*           determine if want default or change in layer                *
*           if want to change layer set layer valid to false            *
*         while layer not valid                                         *
*           Print options;                                              *
*           Get option,                                                 *
*           Check validity.                                             *
*                                                                       *
************************************************************************/
Def_Layer (layer_valid)
int     layer_valid;
{
if (layer_valid == TRUE)
  {
  printf ("\n\n\n\nDo you want the default layer to be %c (ret or N): ",layer);
  if (Get_Option () != '\n')
    layer_valid = FALSE;
  }
while (layer_valid == FALSE)
  {
  printf ("Available layers are: \n\tF = First level metal \n\tS = Second ");
  printf ("level metal \n\tD = Diffusion \n\tP = Polysilicon\n\n");
  printf ("\nSelect a layer (F,S,D,P):");
  layer = Get_Option ();
  if (layer == 'F' || layer == 'S' || layer == 'D' || layer == 'P')
    layer_valid = TRUE ;
    else printf ("\n\n\t Not a valid layer \n\n");
  }
}

#endif
```

```
#ifneed Get_Num
/************************************************************************
*                                                                       *
*       Module Name:  Get_Num                                           *
*       Module Number:  1.3.1.2                                         *
*       Version/Date:  1.0  1 August 1985                               *
*       Input Parameters:  none                                         *
*       Output Parameters:  number                                      *
*       Globals Used: none                                              *
*       Modules Called:  none                                           *
*       Calling Modules:  Def_Wire, Def_Port, Def_Rectangle, Def_Via,   *
*                  Get_Port, Get_Rect, Get_Wire, Get_Via, Get_Call,     *
*                  Get_C_Parameters, Get_Iter, Call_Parameters,         *
*                  Iter_Parameters, M_Implememt, Mod_Port, Mod_Rect,    *
*                  Mod_Wire, Mod_Via                                    *
*       Function:  Get a number input from the user                     *
*       PDL:                                                            *
*         While the incomming character is not a blank, comma,         *
*               semicolon, return, or close parentheses                 *
*           include tne character in the digit string;                  *
*         Add the end of string marker;                                 *
*         Convert the ascii string to an integer;                       *
*         If the input was from the user instead of from disk           *
*           continue getting characters until a return;                 *
*         return the number.                                            *
*                                                                       *
************************************************************************/

Get_Num (fp)

FILE *fp;

{
int     i,c, number;

char    a_digit[16];
i=0;
while ((c=getc(fp)) != ' ' && c != ',' && c != ';' && c != '\n' && c!=')')
  a_digit[i++] = c;
a_digit[i] = '\0';
number = atoi (a_digit);
if (fp == 0 && c != '\n')
  while (getc(fp) != '\n');
return (number);
}
#endif
```

```
#ifneed Port_Tree
/************************************************************************
*       Module Name:  Port_Tree                                         *
*       Module Number:  1.3.1.3                                         *
*       Version/Date:  1.0  7 September 1985                            *
*       Input Parameters:  p_ptr, portname, p_x, p_y                    *
*       Output Parameters:  none                                        *
*       Globals Used: space_used, layer                                 *
*       Modules Called: Port_Tree (recursion)                           *
*       Calling Modules:  Port_Tree, Def_Port, Get_Port                 *
*       Function:  Place a port in the inter_ports structure            *
*       PDL:                                                            *
*         If port location is empty                                     *
*           allocate memory,                                            *
*           place information;                                          *
*         else go to next port location.                                *
*                                                                       *
************************************************************************/

Port_Tree (portname, p_x, p_y, p_ptr)

char    *portname;
struct  inter_ports **p_ptr;

{
int     i;

if (*p_ptr == NULL)
  {
  space_used = space_used + sizeof (struct inter_ports);
  *p_ptr = alloc (sizeof (struct inter_ports));
  (*p_ptr)->next_port = NULL;
  i = 0;
  while (((*p_ptr)->port_name[i] = portname[i]) != '\0') ++i;
  (*p_ptr)->port_layer = layer;
  (*p_ptr)->x_loc = p_x;
  (*p_ptr)->y_loc = p_y;
  }
  else
    Port_Tree (portname, p_x, p_y, &(*p_ptr)->next_port);
}


#endif
```

```
#ifneed Def_Rectangle
/************************************************************************
*                                                                       *
*       Module Name:  Def_Rectangle                                     *
*       Module Number:  1.3.2                                           *
*       Version/Date:  1.0  5 August 1985                               *
*       Input Parameters: none                                          *
*       Output Parameters:  none                                        *
*       Globals Used: layer, f_rect, s_rect, d_rect, p_rect             *
*       Modules Called:  Def_Layer, Rect_Tree, Get_Num, Get_Option      *
*       Calling Modules:  Implement                                     *
*       Function:  Define rectangle (boxes) parameters                  *
*       PDL:                                                            *
*         Define layer;                                                 *
*         While not finished placing rectangles                         *
*           While not finished placing rectangles current default layer *
*             Input height, length, x,y location;                       *
*             Place rectangle in tree structure;                        *
*           If want to place another rectangle in a different layer     *
*             Define layer.                                             *
*                                                                       *
************************************************************************/
Def_Rectangle ()
{
int     finished, fini_layer, r_h, r_l, r_x, r_y;
printf ("\n\n\n\t\t\t READY TO PLACE A RECTANGLE \n\n\n");
Def_Layer (TRUE);
finished = FALSE;
while (finished == FALSE)
  {
  fini_layer = FALSE;
  while (fini_layer == FALSE)
    {
    printf ("\n\nInput Rectangle height,length, and ");
    printf ("x,y location of bottom left corner:\n");
    printf ("\n\t\t Height: ");    r_h = Get_Num ();
    printf ("\n\t\t Length: ");    r_l = Get_Num ();
    printf ("\n\t\t X Location: ");    r_x = Get_Num ();
    printf ("\n\t\t Y Location: ");    r_y = Get_Num ();
    if (layer == 'F')  Rect_Tree (r_h, r_l, r_x, r_y, &f_rect);
    else if (layer == 'S') Rect_Tree (r_h, r_l, r_x, r_y, &s_rect);
    else if (layer == 'D') Rect_Tree (r_h, r_l, r_x, r_y, &d_rect);
    else if (layer == 'P') Rect_Tree (r_h, r_l, r_x, r_y, &p_rect);
    printf ("\n Another Rectangle in this layer (return or N): ");
    if (Get_Option () != '\n') fini_layer = TRUE;
    }
  printf ("\n Another Rectangle in any layer (return or N):");
  if (Get_Option() != '\n')  finished = TRUE;
    else Def_Layer (FALSE);
  }
}
#endif
```

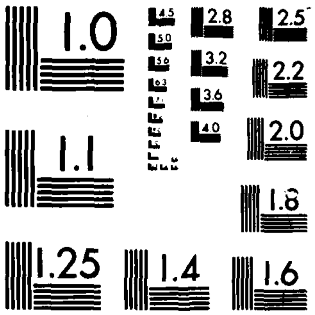
```
#ifneed Rect_Tree
/***********************************************************************
*                                                                      *
*       Module Name:  Rect_Tree                                        *
*       Module Number:  1.3.2.1                                        *
*       Version/Date:  1.0  25 July 1985                               *
*       Input Parameters:  r_h, r_l, r_x, r_y, rect                    *
*       Output Parameters:  none                                       *
*       Globals Used: space_used                                       *
*       Modules Called:  Rect_Tree (recursion)                         *
*       Calling Modules:  Def_Rectangle, Rect_Tree, Get_Rect, Mod_Rect *
*               Out_W_Segment                                          *
*       Function:  Place rectangle in proper position in tree          *
*               based on x location.                                   *
*       PDL:  if rectangle being examined is empty                     *
*               then                                                   *
*                 Allocate Space,                                      *
*                 Place height, length, x,y location in structure;     *
*               else if x to be placed < x examined                    *
*                 then Enter Tree to the left;                         *
*               else if x to be placed = x examined                    *
*                 then Enter Tree to the middle;                       *
*               else if x to be placed > x examined                    *
*                 then Enter Tree to the right.                        *
*                                                                      *
***********************************************************************/

Rect_Tree (r_h, r_l, r_x, r_y, rect)

struct rectangle **rect;
{
if (*rect == NULL)
  {
  space_used = space_used + sizeof (struct rectangle);
  *rect = alloc (sizeof (struct rectangle));
  (*rect)->less_than_x = (*rect)->equals_x = (*rect)->greater_than_x = NULL;
  (*rect)->height = r_h;
  (*rect)->length = r_l;
  (*rect)->x_location = r_x;
  (*rect)->y_location = r_y;
  }
else
  {
  if (r_x < (*rect)->x_location)
    Rect_Tree (r_h, r_l, r_x, r_y, &(*rect)->less_than_x);
  else if (r_x == (*rect)->x_location)
    Rect_Tree (r_h, r_l, r_x, r_y, &(*rect)->equals_x);
  else if (r_x > (*rect)->x_location)
    Rect_Tree (r_h, r_l, r_x, r_y, &(*rect)->greater_than_x);
  }
}
#endif
```

```
#ifneed Def_Wire
/***********************************************************************
*                                                                      *
*       Module Name:  Def_Wire                                         *
*       Module Number:  1.3.3                                          *
*       Version/Date:  1.0  28 July 1985                               *
*       Input Parameters:  none                                        *
*       Output Parameters:  none                                       *
*       Globals Used: layer, seg_ptr, wire_ptr                         *
*       Modules Called:  Def_Layer, Get_Num, S_Wire_Tree, Get_Option   *
*                       Get_W_Option, Place_Wire_Parameters            *
*       Calling Modules:  Implement                                    *
*       Function:  Define a wire similar to CLL                        *
*       PDL:                                                           *
*         While not finished placing wires                             *
*           Input x,y location for start of wire                       *
*           Start the wire definition in memory                        *
*           Get option for next segment of wire                        *
*           while not finished adding segments to this wire            *
*             if option was 'l'                                        *
*               then define layer                                      *
*             else if option was 'w'                                   *
*               then input width                                       *
*             else if option was 'b', 'f', 'u', or 'd'                 *
*               then input length of segment and place wire parameters *
*                                                                      *
***********************************************************************/

Def_Wire ()
{
int     finished,
        w_x,
        w_y,
        width,
        temp_width,
        length;
char    option;

width = 4;
printf ("\n\n\t\t READY TO PLACE A WIRE\n\n");
finished = FALSE;
while (finished == FALSE)
  {
  printf ("\n\n Input x and y location for beginning of wire\n");
  printf ("\n\t\t X Location: ");
  w_x = Get_Num();
  printf ("\n\t\t Y Location: ");
  w_y = Get_Num ();
  S_Wire_Tree (w_x, w_y, &wire_ptr);
  while ((option = Get_W_Option (width)) != 'e')
    {
    if (option == 'l')
      {
      if (layer != 'F')
        {
        printf ("\n Only allowable layer change is to first metal \n");
        printf ("\n Default layer changed to F \n");
        layer = 'F';
        }
        else Def_Layer(FALSE);
      }
    else if (option == 'w')
      {
      printf ("\n Input width or return for default of %d: ",width);
      temp_width = Get_Num ();
      if (temp_width != 0)
      width = temp_width;
      }
    else if (option == 'b' || option == 'f' || option == 'u'
                         || option == 'd')
      {
      printf ("\n\t Input length of this segment: ");
      length = Get_Num ();
      Place_Wire_Parameters (width, length, option, seg_ptr);
      }
    }
  printf ("\n\n Do you want to place another wire (return or N): ");
  if (Get_Option () != '\n') finished = TRUE;
  }
}

#endif
```

```
#ifneed S_Wire_Tree
/************************************************************************
*                                                                       *
*       Module Name:  S_Wire_Tree                                       *
*       Module Number:  1.3.3.1                                         *
*       Version/Date:  1.0  1 August 1985                               *
*       Input Parameters:  w_x, w_y, ptr                                *
*       Output Parameters:  none                                        *
*       Globals Used: space_used                                        *
*       Modules Called:  S_Wire_Tree (recursion)                        *
*       Calling Modules:  Def_Wire, S_Wire_Tree, Get_Wire, Mod_Wire     *
*       Function:  Initialize the memory location for storing the wire  *
*       PDL:                                                            *
*         if the current pointer doen't point to a structure            *
*           then                                                        *
*             allocate enough space for the structure to fit,           *
*             assign the pointer to this space,                         *
*             set the segment pointer,                                  *
*             place the beginning x,y location;                         *
*           else                                                        *
*             if x to be placed < x examined                            *
*               then enter wire tree to the left;                       *
*             else if x to be placed = x examined                       *
*               then enter wire tree to the middle;                     *
*             else if x to be placed > x examined                       *
*               then enter wire tree to the right.                      *
*                                                                       *
************************************************************************/
S_Wire_Tree (w_x, w_y, ptr)
struct wire  **ptr;
{
if (*ptr == NULL)
  {
  space_used = space_used + sizeof (struct wire);
  *ptr = alloc ( sizeof ( struct wire ) );
  (*ptr)->w_l_t_x = (*ptr)->w_e_x = (*ptr)->w_g_t_x = NULL;
  (*ptr)->w_param = NULL;
  seg_ptr = &(*ptr)->w_param;
  (*ptr)->w_x_location = w_x;
  (*ptr)->w_y_location = w_y;
  }
else
  {
  if (w_x < (*ptr)->w_x_location)
    S_Wire_Tree (w_x, w_y, &(*ptr)->w_l_t_x);
  else if (w_x == (*ptr)->w_x_location)
    S_Wire_Tree (w_x, w_y, &(*ptr)->w_e_x);
  else if (w_x > (*ptr)->w_x_location)
    S_Wire_Tree (w_x, w_y, &(*ptr)->w_g_t_x);
  }
}
#endif
```

```
#ifneed Get_W_Option
/************************************************************************
*                                                                       *
*       Module Name:  Get_W_Option                                      *
*       Module Number:  1.3.3.2                                         *
*       Version/Date:  1.0  1 August 1985                               *
*       Input Parameters:  width                                        *
*       Output Parameters:  none                                        *
*       Globals Used:  layer                                            *
*       Modules Called:  none                                           *
*       Calling Modules:  Def_Wire                                      *
*       Function:  Get user's response for placing segments of wires    *
*       PDL:                                                            *
*         while not a valid response                                    *
*           Print options;                                              *
*           Get Response;                                               *
*                                                                       *
************************************************************************/
Get_W_Option (width)

{
int     option_valid;

char    option;


option_valid = FALSE;
while (option_valid == FALSE)
  {
  printf ("\n\n\t\t\t\t Memory used: %d  \n",space_used);
  printf ("\n Do you want to ");
  printf ("\n\t L = change Layers \t default layer = %c", layer);
  printf ("\n\t W = change Width \t default width = %d", width);
  printf ("\n\t B = move Backwards (left) \n\t F = move forwards (right)");
  printf ("\n\t U = move Up \n\t D = move Down ");
  printf ("\n\t E = Exit this wire \n SELECT OPTION: ");
  option = tolower (getc(0));
  if (option != '\n')
    while (getc(0) != '\n');
  if (option == 'l' || option == 'w' || option == 'b' || option == 'f'
    ||option == 'f' || option == 'u' || option == 'd' || option == 'e')
    option_valid = TRUE;
    else printf ("\n Option Not Valid \n");
  }
return (option);
}
#endif
```

AD-A164 109 A MICROCOMPUTER-BASED MENU-DRIVEN CHIP LAYOUT LANGUAGE 3/4
(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH
SCHOOL OF ENGINEERING S C MORRESE DEC 85
UNCLASSIFIED AFIT/GCS/ENG/85D-11 F/G 9/5 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```
#ifneed Place_Wire_Parameters
/************************************************************************
*                                                                       *
*       Module Name:  Place_Wire_Parameters                             *
*       Module Number:  1.3.3.3                                         *
*       Version/Date:  1.0  1 August 1985                               *
*       Input Parameters:  w_w,w_l, dir, ptr_w                          *
*       Output Parameters:  none                                        *
*       Globals Used:  layer, space_used                                *
*       Modules Called:  Place_Wire_Parameters (recursion)              *
*       Calling Modules:  Def_Wire, Place_Wire_Parameters, Get_Wire,    *
*                  Mod_Wire                                             *
*       Function:  Place the segments of wire into memory               *
*       PDL:                                                            *
*         If wire segment being examined is empty                       *
*           allocate memory,                                            *
*           place parameters;                                           *
*         else go to next wire segment.                                 *
*                                                                       *
************************************************************************/

Place_Wire_Parameters (w_w, w_l, dir, ptr_w)

struct wire_parameters  **ptr_w;

{
if (*ptr_w == NULL)
  {
  space_used = space_used + sizeof (struct wire_parameters);
  *ptr_w = alloc (sizeof (struct wire_parameters));
  (*ptr_w)->next_segment = NULL;
  (*ptr_w)->width = w_w;
  (*ptr_w)->length = w_l;
  (*ptr_w)->w_layer = layer;
  (*ptr_w)->direction = dir;
  }
  else
    Place_Wire_Parameters (w_w, w_l, dir, &(*ptr_w)->next_segment);
}

#endif
```

```
#ifneed Def_Via
/************************************************************************
*                                                                       *
*       Module Name:  Def_Via                                           *
*       Module Number:  1.3.4                                           *
*       Version/Date:  1.0  28 July 1985                                *
*       Input Parameters:  none                                         *
*       Output Parameters:  none                                        *
*       Globals Used:  layer, s_via, d_via, p_via                       *
*       Modules Called:  Def_Layer, Via_Tree, Get_Num, Get_Option       *
*       Calling Modules:  Implement                                     *
*       Function:  Define via parameters layer and x,y location         *
*       PDL:  Define Layer;                                             *
*         While not finished placing vias                               *
*           While not finished current layer                            *
*             Input x,y location of via                                 *
*             place via in structure;                                   *
*           If want to place vias in another layer                      *
*             Define Layer;                                             *
************************************************************************/
Def_Via ()
{
int     finished, fini_layer, x_location, y_location;
printf ("\n\n\n\n\n\n\n\n \t\t\t READY TO PLACE A VIA ");
Def_Layer (FALSE);  finished = FALSE;
while (finished == FALSE)
  {
  while (layer == 'F')
    {
    printf ("\n Layer F not valid for vias; choose another\n");
    Def_Layer (FALSE);
    }
  fini_layer = FALSE;
  while (fini_layer == FALSE)
    {
    printf ("\n\n Input via x and y locations\n\n");
    printf ("\n\t\t x location: ");
    x_location = Get_Num ();
    printf ("\n\n\t\t y location: ");
    y_location = Get_Num ();
    if (layer == 'S') Via_Tree (x_location, y_location, &s_via);
    else if (layer == 'D') Via_Tree (x_location, y_location, &d_via);
    else if (layer == 'P') Via_Tree (x_location, y_location, &p_via);
    printf ("\n\n Another via this layer (return or N): ");
    if (Get_Option () != '\n') fini_layer = TRUE;
    }
  printf ("\n\n\n Another via any layer (return or N): ");
  if (Get_Option () != '\n') finished = TRUE;
  else Def_Layer (FALSE);
  }
}
#endif
```

```
#ifneed Via_Tree
/************************************************************************
*                                                                       *
*       Module Name:  Via_Tree                                          *
*       Module Number:  1.3.4.1                                         *
*       Version/Date:  1.0   28 July 1985                               *
*       Input Parameters:  v_x, v_y, l_via                              *
*       Output Parameters:  none                                        *
*       Globals Used:  space_used                                       *
*       Modules Called:  Via_Tree (recursion)                           *
*       Calling Modules:  Def_Via, Via_Tree, Get_Via, Mod_Via,          *
*                  Out_W_Segment                                        *
*       Function:  Place via in proper place by x location              *
*       PDL:                                                            *
*         If via location is empty                                      *
*           then allocate space and place x,y locations in structure;   *
*         else if v_x to be placed < x examined                         *
*           then enter via tree to the left;                            *
*         else if v_x to be placed = x examined                         *
*           then enter via tree to the middle;                          *
*         else if v_x to be placed > x examined                         *
*           then enter via tree to the right.                           *
*                                                                       *
************************************************************************/

Via_Tree (v_x, v_y, l_via)

struct via      **l_via;

{
if (*l_via == NULL)
  {
  space_used = space_used + sizeof (struct via);
  *l_via = alloc (sizeof (struct via));
  (*l_via)->v_l_t_x = (*l_via)->v_e_x = (*l_via)->v_g_t_x = NULL;
  (*l_via)->v_x_location = v_x;
  (*l_via)->v_y_location = v_y;
  }
  else
    {
    if (v_x < (*l_via)->v_x_location)
      Via_Tree (v_x, v_y, &(*l_via)->v_l_t_x);
    else if (v_x == (*l_via)->v_x_location)
      Via_Tree (v_x, v_y, &(*l_via)->v_e_x);
    else if (v_x > (*l_via)->v_x_location)
      Via_Tree (v_x, v_y, &(*l_via)->v_g_t_x);
    }
}
#endif
```

```
#ifneed Def_Flash
/*************************************************************************
*       Module Name:  Def_Flash                                          *
*       Module Number:  1.3.5                                            *
*       Version/Date:  1.0  28 July 1985                                 *
*       Input Parameters:  none                                          *
*       Output Parameters:  none                                         *
*       Globals Used:  none                                              *
*       Modules Called:  none                                            *
*       Calling Modules:  Implement                                      *
*       Function:  Define a round flash based on user inputs             *
*       PDL:  N/A                                                        *
*                                                                        *
*************************************************************************/


Def_Flash ()
{
printf ("\n\n Define Flash is not yet implemented \n\n");
}

#endif
```

```
#ifneed Print_Work
/**********************************************************************
*       Module Name:  Print_Work                                      *
*       Module Number:  1.3.6                                         *
*       Version/Date:  1.0  28 July 1985                              *
*       Input Parameters:  none                                       *
*       Output Parameters:  none                                      *
*       Globals Used:  port_ptr, f_rect, s_rect, d_rect, p_rect, s_via *
*                 d_via, p_via, wire_ptr, call_ptr, iter_ptr          *
*       Modules Called: Get_Option, Walk_P_Tree, Walk_R_Tree,         *
*                 Walk_W_Tree, Walk_Via_Tree, Walk_I_Tree             *
*       Calling Modules:  Implement, D_Implement, M_Implement         *
*       Function:  Print the work accomplished                        *
*       PDL:  N/A                                                     *
*                                                                     *
**********************************************************************/

Print_Work ()

{
char    option;

printf ("\n\n Do you want to see the Ports (ret or N): ");
if (Get_Option () == '\n')
  Walk_P_Tree (port_ptr);


printf ("\n\n Do you want to see the rectangles (ret or N): ");
if (Get_Option () == '\n')
  {
  printf ("\n\n First metal rectangles (ret or N):");
  if (Get_Option () == '\n')
    Walk_R_Tree (f_rect);
  printf ("\n\n Second m rectangles (ret or N):");
  if (Get_Option () == '\n')
    Walk_R_Tree (s_rect);
  printf ("\n\n Rectangles in Diff (ret or N):");
  if (Get_Option () == '\n')
    Walk_R_Tree (d_rect);
  printf ("\n\n Rectangles in Poly (ret or N):");
  if (Get_Option () == '\n')
    Walk_R_Tree (p_rect);
  }


printf ("\n\n Do you want to see the wires (ret or N): ");
if (Get_Option () == '\n')
  Walk_W_Tree (wire_ptr);
printf ("\n\n Do you want to see the vias (ret or N): ");
if (Get_Option () == '\n')
  {
  printf ("\n\n Vias to Second metal (ret or N): ");
  if (Get_Option () == '\n')
    Walk_Via_Tree (s_via);
  printf ("\n\n Vias to Diffusion (ret or N): ");
  if (Get_Option () == '\n')
    Walk_Via_Tree (d_via);
  printf ("\n\n Vias to Poly (ret or N): ");
  if (Get_Option () == '\n')
    Walk_Via_Tree (p_via);
  }


printf ("\n\n Do you want to see the call statements (ret or N): ");
if (Get_Option () == '\n')
  Walk_C_Tree (call_ptr);


printf ("\n\n Do you want to see the iterated statements (ret or N): ");
if (Get_Option () == '\n')
  Walk_I_Tree (iter_ptr);


printf ("\n Hit return to continue: ");
while (getc(0) != '\n');
}


#endif
```

```
#ifneed Walk_P_Tree
/************************************************************************
*                                                                       *
*       Module Name:  Walk_P_Tree                                       *
*       Module Number:  1.3.6.1                                         *
*       Version/Date:  1.0   7 September 1985                           *
*       Input Parameters:  p_ptr                                        *
*       Output Parameters:  none                                        *
*       Globals Used:  none                                             *
*       Modules Called:  Walk_P_Tree (recursion)                        *
*       Calling Modules:  Print_Work, Walk_P_Tree                       *
*       Function:  Walk the port tree                                   *
*       PDL:                                                            *
*         While the location is defined                                 *
*           if the port hasn't been deleted                             *
*             print the port;                                           *
*           Go to the next location.                                    *
*                                                                       *
************************************************************************/

Walk_P_Tree (p_ptr)

struct inter_ports *p_ptr;

{
if (p_ptr != NULL)
  {
  if (p_ptr->y_loc != -9999)
    {
    printf ("\n Port: %s  %c  ", p_ptr->port_name, p_ptr->port_layer);
    printf ("%d,%d\n", p_ptr->x_loc, p_ptr->y_loc);
    }
  Walk_P_Tree (p_ptr->next_port);
  }
}

#endif
```

```
#ifneed Walk_R_Tree
/***********************************************************************
*                                                                      *
*       Module Name:  Walk_R_Tree                                      *
*       Module Number:  1.3.6.2                                        *
*       Version/Date:  1.0   28 July 1985                              *
*       Input Parameters:  rect: pointer to next rectangel             *
*       Output Parameters:  none                                       *
*       Globals Used:  none                                            *
*       Modules Called:  Walk_R_Tree (recursion)                       *
*       Calling Modules:  Print_Work, Walk_R_Tree                      *
*       Function:  Walk the rectangle tree given the layer             *
*       PDL:                                                           *
*         If the location is defined                                   *
*           Walk the left rectangle structure;                         *
*           if the rectangle hasn't been deleted                       *
*             print the rectangle;                                     *
*           Walk the middle rectangle structure;                       *
*           Walk the right rectangle structure.                        *
*                                                                      *
***********************************************************************/

Walk_R_Tree (rect)
struct rectangle *rect;
{
if (rect != NULL)
  {
  Walk_R_Tree (rect->less_than_x);
  if (rect->height != 0 || rect->length != 0)
    {
    printf("\nHeight= %d, length= %d, ",rect->height,rect->length);
    printf("x= %d, y=%d\n",rect->x_location,rect->y_location);
    }
  Walk_R_Tree (rect->equals_x);
  Walk_R_Tree (rect->greater_than_x);
  }
}


#endif
```

```
#ifneed Walk_W_Tree
/**********************************************************************
*                                                                     *
*       Module Name:  Walk_W_Tree                                     *
*       Module Number:  1.3.6.3                                       *
*       Version/Date:  1.0   28 July 1985                             *
*       Input Parameters:  ptr                                        *
*       Output Parameters:  none                                      *
*       Globals Used:  none                                           *
*       Modules Called:  Segment_Tree, Walk_W_Tree (recursion)        *
*       Calling Modules:  Print_Work, Walk_W_Tree                     *
*       Function:  Walk the wire tree given the layer                 *
*       PDL:                                                          *
*         if the location is defined                                  *
*           Walk the left wire tree;                                  *
*           if the wire has segments defined                          *
*             Print the wire start,                                   *
*             Print the wire segments;                                *
*           Walk the middle wire tree;                                *
*           Walk the right wire tree.                                 *
*                                                                     *
**********************************************************************/

Walk_W_Tree (ptr)

struct wire *ptr;

{
if (ptr != NULL)
  {
  Walk_W_Tree (ptr->w_l_t_x);
  if (ptr->w_param != NULL)
    {
    printf("\n Wire x,y = %d,%d",ptr->w_x_location,ptr->w_y_location);
    Segment_Tree (0, ' ', ptr->w_param);
    }
  Walk_W_Tree (ptr->w_e_x);
  Walk_W_Tree (ptr->w_g_t_x);
  }
}

#endif
```

```
#ifneed Segment_Tree
/***********************************************************************
*                                                                      *
*       Module Name:  Segment_Tree                                     *
*       Module Number:  1.3.6.3.1                                      *
*       Version/Date:  1.0   28 July 1985                              *
*       Input Parameters:  p_width, p_layer, seg                       *
*       Output Parameters:  none                                       *
*       Globals Used:  none                                            *
*       Modules Called:  Segment_Tree (recursion)                      *
*       Calling Modules:  Walk_W_Tree, Segment_Tree, Mod_Wire          *
*       Function:  Get the wires' segments                             *
*       PDL:                                                           *
*         If the location is defined                                   *
*           If this segment is on a different layer                    *
*             print the new layer;                                     *
*           If this segment has a different width                      *
*             print the ne width;                                      *
*           Go to the next wire segment.                               *
*                                                                      *
***********************************************************************/

Segment_Tree (p_width, p_layer, seg)

struct wire_parameters *seg;

{
if (seg != NULL)
  {
  if (p_layer != seg->w_layer)
    printf (" %c",seg->w_layer);
  if (p_width != seg->width)
    printf (" W %d", seg->width);
  printf (" %c %d", seg->direction, seg->length);
  Segment_Tree (seg->width, seg->w_layer, seg->next_segment);
  }
}


#endif
```

```
#ifneed Walk_Via_Tree
/************************************************************************
*                                                                       *
*       Module Name:  Walk_Via_Tree                                     *
*       Module Number:  1.3.6.4                                         *
*       Version/Date:  1.0  28 July 1985                                *
*       Input Parameters:  l_via                                        *
*       Output Parameters:  none                                        *
*       Globals Used:  none                                             *
*       Modules Called:  Walk_Via_Tree (recursion)                      *
*       Calling Modules:  Print_Work, Walk_Via_Tree                     *
*       Function:  traverse the via tree give the layer                 *
*       PDL:                                                            *
*         If the location is defined                                    *
*           Walk the left via tree;                                     *
*           if the via has not been deleted                             *
*             print the via;                                            *
*           Walk the middle via tree;                                   *
*           Walk the right via tree;                                    *
*                                                                       *
************************************************************************/

Walk_Via_Tree (l_via)
struct rectangle *l_via;
{
if (l_via != NULL)
  {
  Walk_Via_Tree (l_via->v_l_t_x);
  if (l_via->v_y_location != -9999)
    printf("x= %d, y=%d\n\n",l_via->v_x_location,l_via->v_y_location);
  Walk_Via_Tree (l_via->v_e_x);
  Walk_Via_Tree (l_via->v_g_t_x);
  }
}

#endif
```

```
#ifneed Walk_C_Tree
/************************************************************************
*                                                                       *
*       Module Name:  Walk_C_Tree                                       *
*       Module Number:  1.3.6.5                                         *
*       Version/Date:  1.0  21 Aug 1985                                 *
*       Input Parameters:  ptr                                          *
*       Output Parameters:  none                                        *
*       Globals Used:  c_port_ptr                                       *
*       Modules Called:  Walk_C_Tree (recursion)                        *
*       Calling Modules:  Print_Work, Walk_C_Tree                       *
*       Function:  traverse the call statements tree                    *
*       PDL:                                                            *
*         If the location is defined                                    *
*           Walk the left call tree;                                    *
*           If the call hasn't been deleted                             *
*             print the call statement,                                 *
*             while there are more ports defined                        *
*               if the port hasn't been deleted, print the port;        *
*           Walk the middle call tree;                                  *
*           Walk the right call tree.                                   *
************************************************************************/
Walk_C_Tree (ptr)
struct call_statement  *ptr;
{
if (ptr != NULL)
  {
  Walk_C_Tree (ptr->l_t_name);
  if (ptr->tran_y != -9999)
    {
    printf ("\nC %s %d", ptr->cellname, ptr->cifnum);
    printf (" T %d,%d" ,ptr->tran_x, ptr->tran_y);
    if (ptr->mir_x == TRUE)  printf (" Flip UD");
    if (ptr->mir_y == TRUE)  printf (" Flip LR");
    if (ptr->rot_x != ' ')
      {
      printf (" R %c", ptr->rot_x);
      if (ptr->rot_x == '1') printf ("2");
      }
    c_port_ptr = ptr->c_ports;
    while (c_port_ptr != NULL)
      {
      if (c_port_ptr->port_name[0] != '-')
        printf ("\n\t Port  %s ", c_port_ptr->port_name);
      c_port_ptr = c_port_ptr->next_name;
      }
    }
  Walk_C_Tree (ptr->e_name);
  Walk_C_Tree (ptr->g_t_name);
  }
}
#endif
```

```
#ifneed Walk_I_Tree
/************************************************************************
*                                                                       *
*       Module Name:  Walk_I_Tree                                       *
*       Module Number:  1.3.6.6                                         *
*       Version/Date:  1.0  21 Aug 1985                                 *
*       Input Parameters:  ptr                                          *
*       Output Parameters:  none                                        *
*       Globals Used:  c_port_ptr                                       8
*       Modules Called:  Walk_I_Tree (recursion)                        *
*       Calling Modules:  Print_Work, Walk_I_Tree                       *
*       Function:  traverse the Iterate statement tree                  *
*       PDL:                                                            *
*         If the location is defined                                    *
*           Walk the left iterate tree;                                 *
*           If the iterate hasn't been deleted                          *
*             print the iterate statement,                              *
*             while there are more ports defined print the port;        *
*           Walk the middle iterate tree;                               *
*           Walk the right iterate tree.                                *
************************************************************************/
Walk_I_Tree (ptr)
struct iter_statement *ptr;
{
if (ptr != NULL)
  {
  Walk_I_Tree (ptr->l_t_cifnum);
  if (ptr->x_iters != -9999)
    {
    printf ("\nIterate  %s %d", ptr->cellname, ptr->cifnum);
    printf (" %dx%d ", ptr->x_iters, ptr->y_iters);
    printf ("pitch %d,%d", ptr->x_pitch, ptr->y_pitch);
    printf (" start %d,%d" ,ptr->i_call->tran_x, ptr->i_call->tran_y);
    if (ptr->i_call->mir_x == TRUE)  printf (" Flip UD");
    if (ptr->i_call->mir_y == TRUE)  printf (" Flip LR");
    if (ptr->i_call->rot_x != ' ')
      {
      printf (" R %c", ptr->i_call->rot_x);
      if (ptr->i_call->rot_x == '1') printf ("2");
      }
    c_port_ptr = ptr->i_call->c_ports;
    while (c_port_ptr != NULL)
      {
      if (c_port_ptr->port_name[0] != '-')
        printf ("\n\t Port  %s ", c_port_ptr->port_name);
      c_port_ptr = c_port_ptr->next_name;
      }
    }
  Walk_I_Tree (ptr->e_cifnum);
  Walk_I_Tree (ptr->g_t_cifnum);
  }}
#endif
```

```
#ifneed Save_Cell
/**************************************************************************
*                                                                         *
*       Module Name:  Save_Cell                                           *
*       Module Number:  1.3.7                                             *
*       Version/Date:  1.0  29 August 1985                                *
*       Input Parameters:  filename, cellname                             *
*       Output Parameters:  none                                          *
*       Globals Used:  port_ptr, f_rect, s_rect, d_rect, p_rect, s_via    *
*                  d_via, p_via, wire_ptr, call_ptr, iter_ptr             *
*       Modules Called:  Store_Ptrs, SAVECELL (7.0)                       *
*       Calling Modules:  Implement, D_Implement, M_Implement             *
*       Function:  Call SAVECEll subprogram after storing needed info     *
*       PDL:                                                              *
*         Store the filename;                                             *
*         Store the cellname;                                             *
*         Store the pointers to the structures;                           *
*         Execute SAVECELL.                                               *
*                                                                         *
**************************************************************************/

Save_Cell (filename, cellname)
char    *filename,
        *cellname;
{
int     i, j;
char    a_ptr_argv[80];

i = 0;
while ((a_ptr_argv[i] = filename[i]) != '\0') ++i;
a_ptr_argv[i] = ' ';
j = 0;  ++i;
while ((a_ptr_argv[i] = cellname[j++]) != '\0') ++i;
a_ptr_argv[i] = ' ';
Store_Ptrs (port_ptr, a_ptr_argv, &i);
Store_Ptrs (f_rect, a_ptr_argv, &i);
Store_Ptrs (s_rect, a_ptr_argv, &i);
Store_Ptrs (d_rect, a_ptr_argv, &i);
Store_Ptrs (p_rect, a_ptr_argv, &i);
Store_Ptrs (s_via,  a_ptr_argv, &i);
Store_Ptrs (d_via,  a_ptr_argv, &i);
Store_Ptrs (p_via,  a_ptr_argv, &i);
Store_Ptrs (wire_ptr, a_ptr_argv, &i);
Store_Ptrs (call_ptr, a_ptr_argv, &i);
Store_Ptrs (iter_ptr, a_ptr_argv, &i);
exec ("savecell", a_ptr_argv);
}

#endif
```

```
#ifneed Store_Ptrs
/*************************************************************************
*                                                                        *
*       Module Name:  Store_Ptrs                                         *
*       Module Number:  1.3.7.1                                          *
*       Version/Date:  1.0  9 September 1985                             *
*       Input Parameters:  ptr, a_ptr_argv, i                            *
*       Output Parameters:  a_ptr_argv, i                                *
*       Globals Used:  none                                              *
*       Modules Called:                                                  *
*       Calling Modules:  Save_Cell                                      *
*       Function:  Store the pointers in a string                        *
*       PDL:                                                             *
*         Convert the pointer to an ASCII string;                        *
*         Add it to the existing string of arguments;                    *
*         Insert a blank as a separator between arguments.               *
*                                                                        *
*************************************************************************/

Store_Ptrs (ptr, a_ptr_argv, i)

char    *a_ptr_argv;
int     *i;

{
int     j;
char    a_num [7];

itoa (ptr, a_num);
j = 0;
++*i;
while ((a_ptr_argv[*i] = a_num[j++]) != '\0')
  ++*i;
a_ptr_argv[*i] = ' ';
}

#endif
```

```
/************************************************************************
*                                                                       *
*       Program Name:  DPM.C                                            *
*       Version/Date:  1.0  20 August 1985                              *
*       Function:  This subprogram to MDCLL.C is used to manipulate     *
*               already existing basic cells to arrange them into       *
*               a circuit. It allows the user to place a cell in        *
*               position, move a cell from one location to another,     *
*               or delete a cell from the circuit.                      *
*       Include Files Needed:  MDCLL.H (which includes tprintf.c        *
*               exec.c, and all global variables needed in the          *
*               MDCLL system), DPMLIB.LIB, CRE8LIB.LIB, SAVELIB.LIB,    *
*               and STDLIB.C.                                           *
*       Program Hierarchy:                                              *
*                                                                       *
*       2.0  DPM Main                                                   *
*         |                                                             *
*         |---- 1.1* Name_Cell                                          *
*         |                                                             *
*         |---- 2.1  Get_Cell                                           *
*         |       |                                                     *
*         |       |---- 2.1.1  In_File                                  *
*         |       |                                                     *
*         |       |---- 2.1.2  Get_Port                                 *
*         |       |         |---- 2.1.1* In_File                        *
*         |       |         |---- 1.3.1.2* Get_Num                      *
*         |       |         |---- 1.3.1.3* Port_Tree                    *
*         |       |                                                     *
*         |       |---- 2.1.3  Get_Rect                                 *
*         |       |         |---- 2.1.1* In_File                        *
*         |       |         |---- 1.3.1.2* Get_Num                      *
*         |       |         |---- 1.3.2.1* Rect_Tree                    *
*         |       |                                                     *
*         |       |---- 2.1.4  Get_Wire                                 *
*         |       |         |---- 2.1.1* In_File                        *
*         |       |         |---- 1.3.1.2* Get_Num                      *
*         |       |         |---- 1.3.3.1* S_Wire_Tree                  *
*         |       |         |---- 1.3.3.3* Place_Wire_Parameters        *
*         |       |                                                     *
*         |       |---- 2.1.5  Get_Via                                  *
*         |       |         |---- 2.1.1* In_File                        *
*         |       |         |---- 1.3.1.2* Get_Num                      *
*         |       |         |---- 1.3.4.1* Via_Tree                     *
*         |       |                                                     *
*         |       |---- 2.1.6  Get_Call                                 *
*         |       |         |---- 2.1.1* In_File                        *
*         |       |         |---- 1.3.1.2* Get_Num                      *
*         |       |         |---- 2.1.6.1  Get_C_Parameters             *
*         |       |         |           |---- 1.3.1.2* Get_Num          *
*         |       |         |---- 2.3.1.2* Call_Tree                    *
*         |       |         |---- 2.3.2.3.1* Port_Call_Place            *
*         |       |                                                     *
*         |       |---- 2.1.7  Get_Iter                                   *
*         |                 |---- 2.1.1* In_File                          *
*         |                 |---- 1.3.1.2* Get_Num                        *
*         |                 |---- 2.1.6.1* Get_C_Parameters               *
*         |                 |---- 2.3.3.2* Iter_Tree                      *
*         |                 |---- 2.3.1.2* Call_Tree                      *
*         |                 |---- 2.3.1.3.1* Port_Call_Place              *
*         |                                                               *
*         |---- 1.2.1* Get_Option                                         *
*         |                                                               *
*         |---- 2.2  DPM_Menu                                             *
*         |       |---- 1.2.1* Get_Option                                 *
*         |                                                               *
*         |---- 2.3  D_Implement                                          *
*                 |                                                       *
*                 |---- 1.1* Name_Cell                                    *
*                 |                                                       *
*                 |---- 7.1* Det_Cifnum                                   *
*                 |                                                       *
*                 |---- 1.2.1* Get_Option                                 *
*                 |                                                       *
*                 |---- 2.3.1  Place_Call                                 *
*                 |         |---- 2.3.1.1  Call_Parameters                *
*                 |         |           |---- 1.3.1.2* Get_Num            *
*                 |         |           |---- 1.2.1* Get_Option           *
*                 |         |---- 2.3.1.2r Call_Tree                      *
*                 |         |---- 2.3.1.3  C_Port_Names                   *
*                 |         |           |---- 1.2.1* Get_Option           *
*                 |         |           |---- 2.3.1.3.1r Port_Call_Place*
*                 |         |---- 1.2.1* Get_Option                       *
*                 |                                                       *
*                 |---- 2.3.2r Mod_Call                                   *
*                 |         |---- 1.2.1* Get_Option                       *
*                 |         |---- 2.3.1.1* Call_Parameters                *
*                 |         |---- 2.3.1.3* C_Port_Names                   *
*                 |                                                       *
*                 |---- 2.3.3  Iterate_Call                               *
*                 |         |---- 2.3.1.1* Call_Parameters                *
*                 |         |---- 2.3.3.1  Iter_Parameters                *
*                 |         |           |---- 1.3.1.3* Get_Num            *
*                 |         |---- 2.3.3.2r Iter_Tree                      *
*                 |         |---- 2.3.1.2* Call_Tree                      *
*                 |         |---- 2.3.1.3* C_Port_Names                   *
*                 |                                                       *
*                 |---- 2.3.4r Mod_Iter                                   *
*                 |         |---- 1.2.1* Get_Option                       *
*                 |         |---- 2.3.3.1* Iter_Parameters                *
*                 |         |---- 2.3.1.1* Call_Parameters                *
*                 |         |---- 2.3.1.3* C_Port_Names                   *
*                 |---- 1.3.6* Print_Work                                 *
*                 |---- 1.3.7* Save_Cell                                  *
**************************************************************************/
```

```
#include "mdcll.h"
/*********************************************************************
*                                                                    *
*       Module Name:  Main                                           *
*       Module Number:  2.0                                          *
*       Version/Date:  1.0   21 August 1985                          *
*       Input Parameters:  none                                      *
*       Output Parameters:  none                                     *
*       Globals Used:  none                                          *
*       Modules Called: Get_Cell, DPM_Menu, D_Implement,             *
*                       Name_Cell, Get_Option                        *
*       Calling Modules:  MDCLL program                              *
*       Function:  Delete, Place, or Move a cell in a circuit        *
*       PDL:                                                         *
*         Get the name of the parent cell;                           *
*         If the parent cell is on disk                              *
*           Get the cell;                                            *
*         else if not a new parent cell                              *
*           Parent name not valid so quit;                           *
*         If parent name is valid                                    *
*           While option from DPM menu is other than quit            *
*             Implement the desired option.                          *
*                                                                    *
*********************************************************************/

main ()
{
char    option,
        p_cellname[15],
        p_filename[12];
int     on_disk;

option = ' ';
printf ("\n\n\n\n\n\n\t READY TO DELETE, PLACE, or MOVE A CELL\n\n\n");
printf ("\n\n\n\n\n Input the name of the parent cell: ");
Name_Cell (&on_disk, p_filename, p_cellname);
if (on_disk == TRUE)
  Get_Cell (p_filename, p_cellname);
  else
  {
  printf ("\n Do you want a new parent cell by the name %s",p_cellname);
  printf (" (return or N): ");
  if (Get_Option () != '\n') option = 'Q';
  }
if (option != 'Q')
  while ((option = DPM_Menu (p_cellname)) != 'Q')
    D_Implement (option, p_filename, p_cellname);
}
#include "dpmlib.lib"
#include "cre8lib.lib"
#include "savelib.lib"
#include "stdlib.c"
```

```
#ifneed Get_Cell
/*************************************************************************
*                                                                        *
*       Module Name:  Get_Cell                                           *
*       Module Number:  2.1                                              *
*       Version/Date:  1.0   21 August 1985                              *
*       Input Parameters:  filename, cellname                            *
*       Output Parameters:  none                                         *
*       Globals Used:  none                                              *
*       Modules Called: In_File, Get_Rect, Get_Wire, Get_via,            *
*                       Get_Port, Get_Call, Get_Iter                     *
*       Calling Modules:  DPM Main, MODIFY Main                          *
*       Function:  Get the cell from disk to main memory                 *
*       PDL:                                                             *
*         if file available on disk                                      *
*           while not finished reading                                   *
*             if port definition start then Get Ports                    *
*             else if rectangle start then Get Rectangle                 *
*             else if wire start then Get Wire                           *
*             else if via start then Get Via                             *
*             else if call start then Get Call                           *
*             else if iterate start then Get Iter.                       *
*                                                                        *
* ***********************************************************************/

Get_Cell (filename, cellname)

char    *filename,
        *cellname;

{
int     finished_reading;
char    string[80];
FILE    *fopen(), *fp;

fp = fopen (filename, "r");
if (fp != NULL)
  {
  printf ("\n Cell %s now being read into memory \n", cellname);
  finished_reading = FALSE;
  while (finished_reading == FALSE)
    {
    In_File (fp, '\n', string);
    printf ("%s\n", string);
```

```
 if (string[0] == '(')
   {
   if (string[1] == 'S' && string[2] == 'T' && string[3] == 'A')
    {
    if (string[7] == 'P')
     Get_Port (fp, string);
    else if (string[7] == 'R')
     Get_Rect (fp, string);
    else if (string[7] == 'W')
     Get_Wire (fp, string);
    else if (string[7] == 'V')
     Get_Via  (fp, string);
    else if (string[7] == 'C')
     Get_Call (fp, string);
    else if (string[7] == 'I')
     Get_Iter (fp, string);
    }
   }
   else if (string[0] == 'E')
    finished_reading = TRUE;
  }
 fclose (fp);
 }
}

#endif
```

```
#ifneed In_File
/*************************************************************************
*                                                                        *
*       Module Name:  In_File                                            *
*       Module Number:  2.1.1                                            *
*       Version/Date:  1.0   11 August 1985                              *
*       Input Parameters:  fp, find_char, string                         *
*       Output Parameters:  none                                         *
*       Globals Used:  none                                              *
*       Modules Called:  none                                            *
*       Calling Modules:  Get_Cell, Get_Rect, Get_Wire, Get_Via,         *
*                  Get_Port, Get_Call, Get_Iter, PRIPLO Main, FINALIZE   *
*                  Main, Process_Call, Format_Cif                        *
*       Function:  Read from disk until find_char                        *
*       PDL:                                                             *
*         While input character is not end of file or searched for char *
*           Add the character to the string of input characters.        *
*                                                                        *
*************************************************************************/

In_File (fp, find_char, string)

register FILE  *fp;
register char  find_char;
register char  *string;

{
register char c;
register int  i;

i=0;
while ((c = getc(fp)) != EOF && c != find_char)
  string[i++] = c;
string[i] = '\0';
}
#endif
```

```
#ifneed Get_Port
/*************************************************************************
*                                                                        *
*       Module Name:  Get_Port                                           *
*       Module Number:  2.1.2                                            *
*       Version/Date:  1.0   9 September 1985                            *
*       Input Parameters:  fp, string                                    *
*       Output Parameters:  none                                         *
*       Globals Used:  layer, port_ptr                                   *
*       Modules Called:  In_File, Get_Num, Port_Tree                     *
*       Calling Modules:  Get_Cell                                       *
*       Function:  Read ports from disk into main memory                 *
*       PDL:                                                             *
*         While not finished inputting ports from disk                   *
*           Get a string;                                                *
*           If the input string defines a port                           *
*             Get the port parameters,                                   *
*             Place the port parameters in the port tree;                *
*           else if the string marks the end of the ports                *
*             Set finished to true.                                      *
*                                                                        *
*************************************************************************/
Get_Port (fp, string)
FILE    *fp;
char    *string;
{
int     finished, p_x, p_y;
char    portname [15];
finished = FALSE;
while (finished == FALSE)
  {
  In_File (fp, ' ', string);
  if (string [0] == '9' && string [1] == '4')
    {
    In_File (fp, ' ', portname);
    p_x = Get_Num (fp);
    p_y = Get_Num (fp);
    getc(fp);
    layer = getc(fp);
    if (layer == 'M')
      {
      if (getc(fp) == '2') layer = 'S';
      else layer = 'F';
      }
    In_File (fp, '\n', string);
    Port_Tree (portname, p_x, p_y, &port_ptr);
    }
  else if (string[0] == '(' && string [1] == 'E')
    finished = TRUE;
  }
}
#endif
```

```
#ifneed Get_Rect
/*************************************************************************
*                                                                        *
*       Module Name:  Get_Rect                                           *
*       Module Number:  2.1.3                                            *
*       Version/Date:  1.0   11 August 1985                              *
*       Input Parameters:  fp, string                                    *
*       Output Parameters:  none                                         *
*       Globals Used:  layer, f_rect, s_rect, d_rect, p_rect             *
*       Modules Called:  In_File, Get_Num, Rect_Tree                     *
*       Calling Modules:  Get_Cell                                       *
*       Function:  Read rectangles from disk into main memory            *
*       PDL:                                                             *
*         While not finished getting rectangles                          *
*           If the next input is the end of rectangles marker            *
*             Set finished to true;                                      *
*           else if the next input is a layer definition                 *
*             Set the layer being defined;                               *
*           else if the next input defines a rectangle                   *
*             Get the rectangle parameters,                              *
*             Place the rectangle in memory based on layer.              *
*************************************************************************/
Get_Rect (fp, string)
FILE    *fp;
char    *string;
{
int     rect_finished,  r_h,    r_l,    r_x,    r_y;
rect_finished = FALSE;
while (rect_finished == FALSE)
  {switch (getc(fp))
    {
    case '(' :  In_File (fp, '\n', string);
                if (string[0]=='E' && string[1]=='N' && string[2]=='D')
                  rect_finished = TRUE;
                break;
    case 'L' :  In_File (fp, '\n', string);
                if (string[2] == 'P') layer = 'P';
                else if (string[2] == 'D') layer = 'D';
                else if (string[2] == 'M' && string[3] == '2') layer = 'S';
                else if (string[2] == 'M') layer = 'F';
                break;
    case 'B' :  getc(fp);
                r_h = Get_Num (fp);  r_l = Get_Num (fp);
                r_x = Get_Num (fp) - r_l/2;  r_y = Get_Num (fp) - r_h/2;
                if (layer == 'F') Rect_Tree (r_h,r_l,r_x,r_y,&f_rect);
                else if (layer == 'S') Rect_Tree (r_h,r_l,r_x,r_y,&s_rect);
                else if (layer == 'D') Rect_Tree (r_h,r_l,r_x,r_y,&d_rect);
                else if (layer == 'P') Rect_Tree (r_h,r_l,r_x,r_y,&p_rect);
    }
  }
}
#endif
```

```
#ifneed Get_Wire
/*************************************************************************
*                                                                        *
*       Module Name:  Get_Wire                                           *
*       Module Number:  2.1.4                                            *
*       Version/Date:  1.0   11 August 1985                              *
*       Input Parameters:  fp, string                                    *
*       Output Parameters:  none                                         *
*       Globals Used:  layer, wire_ptr, seg_ptr                          *
*       Modules Called:  In_File, Get_Num, S_Wire_Tree,                  *
*                       Place_Wire_Parameters                            *
*       Calling Modules:  Get_Cell                                       *
*       Function:  Read wires from disk into main memory                 *
*       PDL:                                                             *
*         While not finished reading wires                               *
*           If the next input is an end of wire marker                   *
*             Set finished to true;                                      *
*           else if the next input defines a wire                        *
*             Get the starting x,y location,                             *
*             Start the wire in memory,                                  *
*             While there are more segments                              *
*               Get the segment parameters,                              *
*               Place the parameters in the segment tree.                *
*                                                                        *
*************************************************************************/

Get_Wire (fp,string)

FILE    *fp;
char    *string;

{
int     wire_finished,
        w_x,
        w_y,
        width,
        length;
char    c,
        direction;

wire_finished = FALSE;
while (wire_finished == FALSE)
  {
  if ((c = getc(fp)) == '(')
    {
    In_File (fp, ' ', string);
    if (string[0] == 'E' && string[1] == 'N' && string[2] == 'D')
      {
      In_File (fp, '\n', string);
      wire_finished = TRUE;
      }
      else if (string[0] == 'W' && string[1] == 'i')
          {
          w_x = Get_Num (fp);
          w_y = Get_Num (fp);
          S_Wire_Tree (w_x,w_y,&wire_ptr);
          while ((c = getc(fp)) != ';')
            switch (c)
              {
              case 'F': case 'S': case 'D': case 'P': layer = c; break;
              case 'W': width = Get_Num (fp); break;
              case 'b': case 'f': case 'u': case 'd':
                        direction = c; length = Get_Num (fp);
                        Place_Wire_Parameters (width,length,
                                                    direction,seg_ptr);
              }
           }
    else In_File (fp, '\n', string);
    }
  }
}
#endif
```

```
#ifneed Get_Via
/*************************************************************************
*                                                                        *
*       Module Name:  Get_Via                                            *
*       Module Number:  2.1.5                                            *
*       Version/Date:  1.0   11 August 1985                              *
*       Input Parameters:  fp, string                                    *
*       Output Parameters:  none                                         *
*       Globals Used:  layer, s_via, p_via, d_via                        *
*       Modules Called:  In_File, Get_Num, Via_Tree                      *
*       Calling Modules:  Get_Cell                                       *
*       Function:  Read vias from disk into main memory                  *
*       PDL:                                                             *
*         While not finished reading vias                                *
*           If next input is an end of via marker                        *
*             Set finished to true;                                      *
*           else if next input is the start of a new layer of vias      *
*             While more vias this layer                                 *
*               Get via parameters,                                      *
*               Place parameters in via tree by layer.                   *
*                                                                        *
*************************************************************************/

Get_Via (fp, string)

FILE    *fp;
char    *string;

{
int     via_finished,
        size,
        v_x,
        v_y;
char    c;

via_finished = FALSE;
while (via_finished == FALSE)
  {
  if ((c = getc(fp)) == '(')
    {
    In_File (fp, '\n', string);
    if (string[0] == 'E' && string[1] == 'N' && string[2] == 'D')
      via_finished = TRUE;
      else if (string[8] == 'S' || string[8] == 'D' || string[8] == 'P')
        {
        layer = string[8];
        while ((c = getc(fp)) != '(')
          {
          if (c == 'L')
            In_File (fp, '\n', string);
          else if (c == 'B')
            {
            getc(fp);
            Get_Num (fp);
            size = Get_Num (fp);
            v_x = Get_Num (fp) - size/2;
            v_y = Get_Num (fp) - size/2;
            if (layer == 'S')
              Via_Tree (v_x, v_y, &s_via);
            else if (layer == 'D')
              Via_Tree (v_x, v_y, &d_via);
            else if (layer == 'P')
              Via_Tree (v_x, v_y, &p_via);
            }
          }
        }
    }
    else In_File (fp, '\n', string);
  }
}
#endif
```

```
#ifneed Get_Call
/***********************************************************************
*                                                                      *
*       Module Name:  Get_Call                                         *
*       Module Number:  2.1.6                                          *
*       Version/Date:  1.0   21 August 1985                            *
*       Input Parameters:  fp, string                                  *
*       Output Parameters:  none                                       *
*       Globals Used:  call_ptr, c_port_ptr                            *
*       Modules Called:  In_File, Get_Num, Call_Tree, Get_C_Parameters *
*                  Port_Call_Place                                     *
*       Calling Modules:  Get_Cell, FINALIZE Main                      *
*       Function:  Read calls from disk into main memory               *
*       PDL:                                                           *
*         While not finished getting call statements                   *
*           If next input is end of call marker                        *
*             Set finished to false;                                   *
*           else if next input is a call statement                     *
*             Get call parameters,                                     *
*             Place parameters in call tree structure;                 *
*           else if next input is a portname                           *
*             Get the portname,                                        *
*             Place the port with the call.                            *
*                                                                      *
***********************************************************************/

Get_Call (fp, string)

FILE    *fp;
char    *string;

{
int     cifnum, call_finished,
        t_x,t_y,m_x,m_y,
        min_x, min_y, max_x, max_y;

char    c, r_x, cellname[15], portname[15];

call_finished = FALSE;
while (call_finished == FALSE)
  {
  In_File (fp, ' ', string);
  if (string[1] == 'E' && string[2] == 'N' && string[3] == 'D')
    {
    In_File (fp, '\n', string);
    call_finished = TRUE;
    }
  else if (string[1] == 'C')
    {
    In_File (fp, ' ', cellname);
    t_x = Get_Num (fp);
    t_y = Get_Num (fp);
    In_File (fp, ' ', string);
    min_x = Get_Num (fp);
    min_y = Get_Num (fp);
    max_x = Get_Num (fp);
    max_y = Get_Num (fp);
    In_File (fp, '\n', string);
    }
  else if (string[0] == 'C')
    {
    Get_C_Parameters (&cifnum, &m_x, &m_y, &r_x, fp);
    Call_Tree (cellname, cifnum, t_x,t_y,m_x,m_y,r_x,
                min_x, min_y, max_x, max_y, &call_ptr);
    }
  else if (string[0] == '(' && string[1] == 'P')
    {
    In_File (fp, ')', portname);
    In_File (fp, '\n', string);
    Port_Call_Place (portname, c_port_ptr);
    }
  }
}
#endif
```

```
#ifneed Get_C_Parameters
/************************************************************************
*                                                                       *
*       Module Name:  Get_C_Parameters                                  *
*       Module Number:  2.1.6.1                                         *
*       Version/Date:  1.0   17 September 1985                          *
*       Input Parameters:  fp                                           *
*       Output Parameters:  cifnum, m_x, m_y, r_x                       *
*       Globals Used:  none                                             *
*       Modules Called:  Get_Num                                        *
*       Calling Modules:  Get_Cell, FINALIZE Main                       *
*       Function:  Read calls parameters from disk into main memory     *
*       PDL:                                                            *
*         Get the cifnum;                                               *
*         While not end of current line                                 *
*           If next param is translation                                *
*             Get the translated position;                              *
*           else if the next param is a mirroring                       *
*             Get the mirror in x or y;                                 *
*           else if the next param is a rotation                        *
*             Interpret the rotation.                                   *
*                                                                       *
************************************************************************/

Get_C_Parameters (cifnum, m_x, m_y, r_x, fp)

int     *cifnum, *m_x, *m_y;
char    *r_x;
FILE    *fp;

{
char    c;

*cifnum = Get_Num(fp);
*m_x = *m_y = 0;
*r_x = ' ';

while ((c = getc(fp)) != '\n')
  {
  if (c == 'T')
    {
    getc (fp);
    Get_Num (fp);
    Get_Num (fp);
    }
  else if (c == 'M')
    {
    getc (fp);
    if (getc (fp) == 'X')
      *m_x = TRUE;
    else
      *m_y = TRUE;
    }
  else if (c == 'R')
    {
    getc (fp);
    if ((c = getc (fp)) == '1')
      *r_x = '3';
    else if (c == '-')
      *r_x = '9';
    else if (c == '0')
      {
      getc (fp);
      if (getc (fp) == '1')
        *r_x = '1';
      else
        *r_x = '6';
      }
    }
  }
}
#endif
```

```
#ifneed Get_Iter
/************************************************************************
*                                                                       *
*       Module Name:  Get_Iter                                          *
*       Module Number:  2.1.7                                           *
*       Version/Date:  1.0   17 September 1985                          *
*       Input Parameters:  fp, string                                   *
*       Output Parameters:  none                                        *
*       Globals Used:  c_port_ptr, iter_ptr, i_call_ptr                 *
*       Modules Called:  In_File, Get_Num, Call_Tree, Get_C_Parameters  *
*                  Iter_Tree, Port_Call_Place                           *
*       Calling Modules:  Get_Cell                                      *
*       Function:  Read iters from disk into main memory                *
*       PDL:                                                            *
*         While not finished getting iter statements                    *
*           If next input is end of iter marker                         *
*             Set finished to false;                                    *
*           else if next input is a iter statement                      *
*             Get call parameters,                                      *
*             Place parameters in call tree structure;                  *
*           else if next input is a portname                            *
*             Get the portname,                                         *
*             Place the port with the call.                             *
*                                                                       *
************************************************************************/
Get_Iter (fp, string)
FILE    *fp;
char    *string;
{
int     cifnum, finished,
        n_x, n_y, x_p, y_p,
        t_x, t_y, m_x, m_y,
        min_x, min_y, max_x, max_y;

char    c, r_x, cellname[15], portname[15];

finished = FALSE;
while (finished == FALSE)
  {
  In_File (fp, ' ', string);
  if (string[1] == 'E' && string[2] == 'N' && string[3] == 'D')
    {
    In_File (fp, '\n', string);
    finished = TRUE;
    }
```

```
 else if (string[1] == 'I')
  {
  In_File (fp, ' ', cellname);
  n_x = Get_Num (fp);  n_y = Get_Num (fp);
  x_p = Get_Num (fp);  y_p = Get_Num (fp);
  t_x = Get_Num (fp);  t_y = Get_Num (fp);
  min_x = Get_Num (fp);
  min_y = Get_Num (fp);
  max_x = Get_Num (fp);
  max_y = Get_Num (fp);
  In_File (fp, '\n', string);
  while (getc(fp) != 'C');
  getc(fp);
  Get_C_Parameters (&cifnum, &m_x, &m_y, &r_x, fp);
  Iter_Tree (cellname, cifnum, n_x, n_y, x_p, y_p, &iter_ptr);
  Call_Tree (cellname, cifnum, t_x, t_y, m_x, m_y, r_x,
        min_x, min_y, max_x, max_y, i_call_ptr);
  }
 else if (string[1] == 'P')
  {
  In_File (fp, ')', portname);
  In_File (fp, '\n', string);
  Port_Call_Place (portname, c_port_ptr);
  }
 else In_File (fp, '\n', string);
 }
}
#endif
```

```
#ifneed DPM_Menu
/***********************************************************************
*                                                                      *
*       Module Name:  DPM_Menu                                         *
*       Module Number:  2.2                                            *
*       Version/Date:  1.0   26 August 1985                            *
*       Input Parameters:  p_cellname                                  *
*       Output Parameters:  none                                       *
*       Globals Used:  space_used, AVAIL_SPACE                         *
*       Modules Called:  Get_Option                                    *
*       Calling Modules:  Main (2.0)                                   *
*       Function:  Display options available for dpm                   *
*       PDL:                                                           *
*         While not a valid option                                     *
*           Display options,                                           *
*           Get desired option;                                        *
*         Return option.                                               *
*                                                                      *
***********************************************************************/
DPM_Menu (p_cellname)

char    *p_cellname;

{
char    option;
int     option_valid,
        space_left;

option_valid = FALSE;
while (option_valid == FALSE)
  {
  space_left = AVAIL_SPACE - space_used;
  printf ("\n\n\t\t\t\t Memory space left: %d\n", space_left);
  printf ("\n\n\n\n Options available for cell %s are \n\n", p_cellname);
  printf ("\n   C = place Call statement \t P = Print work \n");
  printf ("\n   I = Iterate a call        \t E = Exit to main menu \n");
  printf ("\n   M = Modify/Delete a call \t     after saving cell \n");
  printf ("\n   D = Delete/Modify an      \t Q = Quit to system \n");
  printf ("\n         Iterate statment \n");
  printf ("\n\n\n SELECT OPTION (C,I,M,D,P,E,Q): ");
  option = Get_Option ();
  if (option == 'C' || option == 'I' || option == 'M' || option == 'D'
    ||option == 'P' || option == 'E' || option == 'Q')
    option_valid = TRUE;
    else  printf ("\n\n Option %c not valid\n", option);
  }
return (option);
}

#endif
```

```
#ifneed D_Implement
/************************************************************************
*                                                                       *
*       Module Name: D_Implement                                        *
*       Module Number:  2.3                                             *
*       Version/Date:  1.0   21 August 1985                             *
*       Input Parameters:  option, p_filename, p_cellname               *
*       Output Parameters:  none                                        *
*       Globals Used:  call_ptr, iter_ptr                               *
*       Modules Called:  Name_Cell, Det_Cifnum, Get_Option, Place_Call, *
*                  Mod_Call, Iterate_Call, Mod_Iter, Print_Work,        *
*                  Save_Cell                                            *
*       Calling Modules:  DPM Main                                      *
*       Function:  Implement the desired option                         *
*       PDL:                                                            *
*         If option is to place, iterate, modify, or delete calls       *
*           Name the cell to be used;                                   *
*           Determine the cifnum;                                       *
*           If option is to place or iterate a call statement           *
*             If file on disk                                           *
*               Get the bounds of this cell,                            *
*             else check if this cell name correct to place,            *
*             If file on disk or place is correct                       *
*               If option is to place                                   *
*                 Place a call statement;                               *
*               else if option is to iterate                            *
*                 Iterate a call statement;                             *
*           else if option is to modify or delete a call statement      *
*             Print Instructions,                                       *
*             Modify the call statements;                               *
*           else if option is to print the work                         *
*             Print the work;                                           *
*           else if option is to save the cell                          *
*             Save the cell to disk.                                    *
*                                                                       *
************************************************************************/
D_Implement (option, p_filename, p_cellname)
char    option,
        *p_filename,
        *p_cellname;
{
int     on_disk, cifnum,
        min_x, min_y, max_x, max_y,
        place, *exit, finished;
char    filename [12], cellname [15], string [80];
FILE    *fopen(), *fp;

if (option == 'C' || option == 'I' || option == 'M' || option == 'D')
  {
  printf ("\n Input name of cell : ");
  Name_Cell (&on_disk, filename, cellname);
  cifnum = Det_Cifnum (cellname);
  if (option == 'C' || option == 'I')
    {
    if (on_disk == TRUE)
      {
      finished = FALSE;
      fp = fopen (filename, "r");
      while (finished == FALSE)
        {
        while (getc(fp) != '(') In_File (fp, '\n', string);
        if (getc(fp) == 'b' && getc(fp) == 'o' && getc(fp) == 'u')
          {
          In_File (fp, ' ', string);
          In_File (fp, ' ', string);
          min_x = Get_Num (fp);  min_y = Get_Num (fp);
          max_x = Get_Num (fp);  max_y = Get_Num (fp);
          finished = TRUE;
          }
        }
      fclose (fp);
      }
      else
      {
      printf ("\n Cell %s not found under filename %s\n",cellname,filename);
      printf ("\n Place a cell by this name anyway (ret or N): ");
      place = Get_Option ();
      }
    if (on_disk == TRUE || place == '\n')
      {
      if (option == 'C')
        Place_Call (min_x, min_y, max_x, max_y, cifnum, cellname);
      else if (option == 'I')
        Iterate_Call (min_x, min_y, max_x, max_y, cifnum, cellname);
      }
    }
  else if (option == 'M' || option == 'D')
    {
    printf ("Type return for next call,E to exit, M to mod,
                         or D to Delete\n");
    *exit = FALSE;
    if (option == 'M')
      Mod_Call (exit, cifnum, call_ptr);
    else Mod_Iter (exit, cifnum, iter_ptr);
    }
  }
  else if (option == 'P')
    Print_Work ();
  else if (option == 'E')
    Save_Cell (p_filename, p_cellname);
}

#endif
```

```
#ifneed Place_Call
/************************************************************************
*                                                                       *
*       Module Name:  Place_Call                                        *
*       Module Number:  2.3.1                                           *
*       Version/Date:  1.0   26 August 1985                             *
*       Input Parameters:  min_x, min_y, max_x, max_y, cifnum, cellname *
*       Output Parameters:  none                                        *
*       Globals Used:  call_ptr                                         *
*       Modules Called:  Get_Option, Call_Parameters, C_Port_Names,     *
*                       Call_Tree                                       *
*       Calling Modules:  DPM Main (2.0)                                *
*       Function: Place a call statement in the cell                    *
*       PDL:                                                            *
*         While not finished placing calls to this cell                 *
*           Determine the parameters for the call;                      *
*           Place the parameters in the tree structure;                 *
*           Place any associated port names.                            *
*                                                                       *
************************************************************************/
Place_Call (min_x, min_y, max_x, max_y, cifnum, cellname)

int     min_x, min_y, max_x, max_y, cifnum;
char    *cellname;

{
int     i, finished_place, t_x, t_y, m_x, m_y;
char    r_x, portname [15];

finished_place = FALSE;
while (finished_place == FALSE)
  {
  Call_Parameters (cellname, &t_x, &t_y, &m_x, &m_y, &r_x);
  Call_Tree (cellname, cifnum, t_x, t_y, m_x, m_y, r_x,
                min_x, min_y, max_x, max_y, &call_ptr);
  C_Port_Names ();
  printf ("\n Place cell %s in another location (ret or N): ", cellname);
  if (Get_Option () != '\n')
    finished_place = TRUE;
  else finished_place = FALSE;
  }
}
#endif
```

```
#ifneed Call_Parameters
/************************************************************************
*                                                                       *
*       Module Name:  Call_Parameters                                   *
*       Module Number:  2.3.1.1                                         *
*       Version/Date:  1.0   26 August 1985                             *
*       Input Parameters:  cellname                                     *
*       Output Parameters:  t_x, t_y, m_x, m_y, r_x                     *
*       Globals Used:  none                                             *
*       Modules Called:  Get_Option, Get_Num                            *
*       Calling Modules:  Place_Call, Iterate_Call, Mod_Call, Mod_Iter  *
*       Function: Get the call parameters                               *
*       PDL:                                                            *
*         Get x,y location for bottom left corner of called cell;       *
*         Determine if cell should be mirrored in x and/or y;           *
*         Determine position of x axis of called cell.                  *
*                                                                       *
************************************************************************/

Call_Parameters (cellname, t_x, t_y, m_x, m_y, r_x)

char    *cellname, *r_x;
int     *t_x, *t_y, *m_x, *m_y;

{
  printf ("\n Enter x and y location for bottom left of cell %s\n",cellname);
  printf ("\n\t X location : ");
  *t_x = Get_Num (0);
  printf ("\n\t Y location : ");
  *t_y = Get_Num (0);
  printf ("\n Do you want this cell flipped upside down (ret or N): ");
  if (Get_Option () == '\n')
    *m_x = TRUE;
    else *m_x = FALSE;
  printf ("\n\t\t Flipped left to right  (ret or N): ");
  if (Get_Option () == '\n')
    *m_y = TRUE;
    else *m_y = FALSE;
  printf ("\n Do you want to rotate the x axis (ret or N): ");
  if (Get_Option () == '\n')
    {
    printf ("\n\t To 3, 6, 9, or 12 o'clock (3,6,9,12): ");
    while ((*r_x = Get_Option ()) != '3' && *r_x != '6'
                && *r_x != '9' && *r_x != '1')
      {
      printf ("\n Option %c not valid; try again\n", *r_x);
      printf ("\n\t To 3, 6, 9, or 12 o'clock (3,6,9,12): ");
      }
    }
    else *r_x = ' ';
}
#endif
```

```
#ifneed Call_Tree
/*************************************************************************
*                                                                        *
*       Module Name:  Call_Tree                                          *
*       Module Number:  2.3.1.2                                          *
*       Version/Date:  1.0   21 August 1985                              *
*       Input Parameters:  name, cifnum, t_x, t_y, m_x, m_y, r_x,        *
*                       ptr, min_x, min_y, max_x, max_y                  *
*       Output Parameters:  none                                         *
*       Globals Used:  space_used, c_port_ptr                            *
*       Modules Called:  Call_Tree (recursion)                           *
*       Calling Modules:  Get_Call, Call_Tree, Place_Call, Iterate_Call *
*       Function:  places calls into main memory                         *
*       PDL:                                                             *
*         If the location being examined is empty                        *
*           Allocate memory,                                             *
*           Place call parameters in structure;                          *
*         else                                                           *
*           If cifnum to be placed < present cifnum                      *
*             Go to the next call to the left;                           *
*           else if cifnum to be placed = present cifnum                 *
*             Go to the next call in the middle;                         *
*           else if cifnum to be placed > present cifnum                 *
*             Go to the next call to the right.                          *
*                                                                        *
*************************************************************************/

Call_Tree (name, cifnum,t_x,t_y,m_x,m_y,r_x,min_x,min_y,max_x,max_y,ptr)

char    *name;
struct call_statement **ptr;

{
int     i;

if (*ptr == NULL)
  {
  space_used = space_used + sizeof (struct call_statement);
  *ptr = alloc (sizeof(struct call_statement));
  (*ptr)->l_t_name = (*ptr)->e_name = (*ptr)->g_t_name = NULL;
  i = 0;
  while (((*ptr)->cellname[i] = name[i]) != '\0')
    ++i;
  (*ptr)->cifnum = cifnum;
  (*ptr)->tran_x = t_x;
  (*ptr)->tran_y = t_y;
  (*ptr)->mir_x  = m_x;
  (*ptr)->mir_y  = m_y;
  (*ptr)->rot_x  = r_x;
  (*ptr)->x_min = min_x;
  (*ptr)->y_min = min_y;
  (*ptr)->x_max = max_x;
  (*ptr)->y_max = max_y;
  (*ptr)->c_ports = NULL;
  c_port_ptr = &(*ptr)->c_ports;
  }
  else
    {
    if (cifnum < (*ptr)->cifnum)
      Call_Tree (name, cifnum, t_x, t_y, m_x, m_y, r_x,
                min_x, min_y, max_x, max_y, &(*ptr)->l_t_name);
    else if (cifnum == (*ptr)->cifnum)
      Call_Tree (name, cifnum, t_x, t_y, m_x, m_y, r_x,
                min_x, min_y, max_x, max_y, &(*ptr)->e_name);
    else if (cifnum > (*ptr)->cifnum)
      Call_Tree (name, cifnum, t_x, t_y, m_x, m_y, r_x,
                min_x, min_y, max_x, max_y, &(*ptr)->g_t_name);
    }
}
#endif
```

```
#ifneed C_Port_Names
/*************************************************************************
*                                                                        *
*       Module Name:  C_Port_Names                                       *
*       Module Number:  2.3.1.3                                          *
*       Version/Date:  1.0   26 August 1985                              *
*       Input Parameters:  none                                          *
*       Output Parameters:  none                                         *
*       Globals Used:  c_port_ptr                                        *
*       Modules Called:  Get_Option, Port_Call_Place                     *
*       Calling Modules:  Place_Call, Iterate_Call, Mod_Call, Mod_Iter  *
*       Function: Get the call parameters                                *
*       PDL:                                                             *
*         If the user wants to place portnames                           *
*           While not finished placing ports                             *
*             Get the portname;                                          *
*             Place the port name with the call.                         *
*                                                                        *
*************************************************************************/

C_Port_Names ()

{
int     i, finished;
char    portname [15];

finished = FALSE;
printf ("\n Do you want to place any portnames (ret or n): ");
if (Get_Option () == '\n')
  while (finished == FALSE)
    {
    printf ("\n Enter port name: ");
    i = 0;
    while ((portname[i] = getc(0)) != '\n')
      ++i;
    portname[i] = '\0';
    Port_Call_Place (portname, c_port_ptr);
    printf ("\n another port name (ret or N): ");
    if (Get_Option () != '\n')
      finished = TRUE;
    }
}
#endif
```

```
#ifneed Port_Call_Place
/*************************************************************************
*                                                                        *
*       Module Name:  Port_Call_Place                                    *
*       Module Number:  2.3.1.3.1                                        *
*       Version/Date:  1.0   10 September 1985                           *
*       Input Parameters:  portname, c_p_ptr                             *
*       Output Parameters:  none                                         *
*       Globals Used:  space_used                                        *
*       Modules Called:  Port_Call_Place (recursion)                     *
*       Calling Modules:  Get_Call, C_Port_Names, Port_Call_Place        *
*       Function:  places calls ports into main memory                   *
*       PDL:                                                             *
*         If the present port location is empty                         *
*           Allocate memory;                                             *
*           Place the portname in the structure;                         *
*         else                                                           *
*           Go to the next port name location.                           *
*                                                                        *
*************************************************************************/
Port_Call_Place (portname, c_p_ptr)

char    *portname;
struct named_ports **c_p_ptr;

{
int     i;

if (*c_p_ptr == NULL)
  {
  space_used = space_used + sizeof (struct named_ports);
  *c_p_ptr = alloc (sizeof (struct named_ports));
  i = 0;
  while (((*c_p_ptr)->port_name[i] = portname[i]) != '\0')
    ++i;
  (*c_p_ptr)->next_name = NULL;
  }
else
  Port_Call_Place (portname, &(*c_p_ptr)->next_name);
}

#endif
```

```
#ifneed Mod_Call
/************************************************************************
*                                                                       *
*       Module Name:  Mod_Call                                          *
*       Module Number:  2.3.2                                           *
*       Version/Date:  1.0   21 August 1985                             *
*       Input Parameters:  exit, cifnum, ptr                            *
*       Output Parameters:  exit                                        *
*       Globals Used:  c_port_ptr                                       *
*       Modules Called:  Get_Option, Call_Parameters, C_Port_Names      *
*       Calling Modules:  D_Implement                                   *
*       Function:  Modify a call statement                              *
*       PDL:                                                            *
*         If present location is not empty and exit is false            *
*           Walk the call tree to the left;                             *
*           if exit is false and cifnum is valid                        *
*             Print the call statement,                                 *
*             Determine option,                                         *
*             If option is to exit                                      *
*               Set exit to true;                                       *
*             else if option is to modify and mod is to delete          *
*               Delete this call statement;                             *
*             else if option is to modify and mod is to modify          *
*               Get and place new call parameters;                      *
*           If exit is false and there are ports defined                *
*             Delete or modify ports as needed;                         *
*           If exit is false                                            *
*             Walk the middle call tree;                                *
*           If exit is false                                            *
*             Walk the right call tree.                                 *
*                                                                       *
************************************************************************/
Mod_Call (exit, cifnum, ptr)

int     *exit, cifnum;
struct call_statement *ptr;
{
char    option, r_x, portname[15];
int     i, exit_ports, finished;

if (ptr != NULL && *exit == FALSE)
  {
  Mod_Call (exit, cifnum, ptr->l_t_name);
  if (*exit == FALSE && cifnum <= ptr->cifnum)
    {
    exit_ports = FALSE;
    printf ("%s %d, %d ", ptr->cellname, ptr->tran_x, ptr->tran_y);
    if (ptr->mir_x)  printf ("Flip UD ");
    if (ptr->mir_y)  printf ("Flip LR ");
  if (ptr->rot_x != ' ')
   {
   printf ("R %c", ptr->rot_x);
   if (ptr->rot_x == '1')
    printf ("2");
   }
  printf (" (ret, E, M, or D): ");
  option = Get_Option ();
  if (option == 'E')
   *exit = TRUE;
  else if (option == 'D')
   {
   exit_ports = TRUE;
   ptr->c_ports = NULL;
   ptr->tran_y = -9999;
   }
  else if (option == 'M')
   Call_Parameters (ptr->cellname, &ptr->tran_x, &ptr->tran_y,
         &ptr->mir_x, &ptr->mir_y, &ptr->rot_x);
  c_port_ptr = ptr->c_ports;
  if (c_port_ptr != NULL && *exit == FALSE)
  printf ("\nType ret for next port, E to Exit, M to Mod,
            or D to Delete\n");
  while (*exit == FALSE && exit_ports == FALSE && c_port_ptr != NULL)
   {
   printf ("\n Port %s (ret, E, M, or D): ", c_port_ptr->port_name);
   option = Get_Option ();
   if (option == 'E')
    exit_ports = TRUE;
   else if (option == 'D')
    c_port_ptr->port_name[0] = '-';
   else if (option == 'M')
    {
    printf ("\n New port name : ");
    i = 0;
    while ((c_port_ptr->port_name[i] = getc(0)) != '\n')  ++i;
    c_port_ptr->port_name[i] = '\0';
    }
   c_port_ptr = c_port_ptr->next_name;
   }
  c_port_ptr = &ptr->c_ports;
  if (*exit == FALSE && exit_ports == FALSE)
   C_Port_Names ();
  }
  if (*exit == FALSE)
   Mod_Call (exit, cifnum, ptr->e_name);
  if (*exit == FALSE)
   Mod_Call (exit, cifnum, ptr->g_t_name);
 }
}
#endif
```

```
#ifneed Iterate_Call
/***********************************************************************
*                                                                      *
*       Module Name:  Iterate_Call                                     *
*       Module Number:  2.3.3                                          *
*       Version/Date:  1.0   21 August 1985                            *
*       Input Parameters:  min_x, min_y, max_x, max_y, cifnum, cellname *
*       Output Parameters:  none                                       *
*       Globals Used:  iter_ptr, i_call_ptr                            *
*       Modules Called:  Call_Parameters, Iter_Parameters, Iter_Tree,  *
*                  Call_Tree, C_Port_Names                             *
*       Calling Modules:  D_Implement                                  *
*       Function:  Get the iteration information from the user         *
*       PDL:  N/A                                                      *
*                                                                      *
***********************************************************************/
Iterate_Call (min_x, min_y, max_x, max_y, cifnum, cellname)

char    *cellname;

{
int     n_x, n_y,
        x_p, y_p,
        t_x, t_y, m_x, m_y, r_x;

Call_Parameters (cellname, &t_x, &t_y, &m_x, &m_y, &r_x);
Iter_Parameters (&n_x, &n_y, &x_p, &y_p, min_x, min_y, max_x, max_y);
Iter_Tree (cellname, cifnum, n_x, n_y, x_p, y_p, &iter_ptr);
Call_Tree (cellname, cifnum, t_x, t_y, m_x, m_y, r_x, min_x, min_y,
                max_x, max_y, i_call_ptr);
C_Port_Names ();
}

#endif
```

```
#ifneed Iter_Parameters
/************************************************************************
*                                                                       *
*       Module Name:  Iter_Parameters                                   *
*       Module Number:  2.3.3.1                                         *
*       Version/Date:  1.0   17 September 1985                          *
*       Input Parameters:  min_x, min_y, max_x, max_y                   *
*       Output Parameters:  n_x, n_y, x_p, y_p                          *
*       Globals Used:  none                                             *
*       Modules Called:  Get_Num                                        *
*       Calling Modules:  Iterate_Call, Mod_Iter                        *
*       Function:  Get the necessary parameters for iterating a cell    *
*       PDL:  N/A                                                       *
*                                                                       *
************************************************************************/
Iter_Parameters (n_x, n_y, x_p, y_p, min_x, min_y, max_x, max_y)

int     *n_x, *n_y, *x_p, *y_p;

{
printf ("\n Input the number of iterations in the x direction: ");
*n_x = Get_Num (0);
printf ("\n\t\t Number in the y direction: ");
*n_y = Get_Num (0);
printf ("\n Input the x pitch or return for bounding box default: ");
*x_p = Get_Num (0);
if (*x_p == 0)
  *x_p = max_x - min_x;
printf ("\n\t\t Y pitch or return for default: ");
*y_p = Get_Num (0);
if (*y_p == 0)
  *y_p = max_y - min_y;
}

#endif
```

```
#ifneed Iter_Tree
/***************************************************************************
*       Module Name:  Iter_Tree                                           *
*       Module Number:  2.3.3.2                                           *
*       Version/Date:  1.0   21 August 1985                               *
*       Input Parameters:  name, cifnum, t_x, t_y, m_x, m_y,r_x,r_y,ptr *
*       Output Parameters:  none                                          *
*       Globals Used:  space_used                                         *
*       Modules Called:  Iter_Tree (recursion)                            *
*       Calling Modules:  Iterate_Cell, Call_Tree                         *
*       Function:  places iterates into main memory                       *
*       PDL:                                                              *
*         If the location being examined is empty                         *
*           Allocate memory,                                              *
*           Place call parameters in structure;                           *
*         else                                                            *
*           If cifnum to be placed < present cifnum                       *
*             Go to the next call to the left;                            *
*           else if cifnum to be placed = present cifnum                  *
*             Go to the next call in the middle;                          *
*           else if cifnum to be placed > present cifnum                  *
*             Go to the next call to the right.                           *
**************************************************************************/
Iter_Tree (name, cifnum, n_x, n_y, x_p, y_p, ptr)
char    *name;
struct iter_statement **ptr;
{
int     i;
if (*ptr == NULL)
  {
  space_used = space_used + sizeof (struct iter_statement);
  *ptr = alloc (sizeof(struct iter_statement));
  (*ptr)->l_t_cifnum = (*ptr)->e_cifnum = (*ptr)->g_t_cifnum = NULL;
  i = 0;
  while (((*ptr)->cellname[i] = name[i]) != '\0')  ++i;
  (*ptr)->cifnum = cifnum;
  (*ptr)->x_iters = n_x;  (*ptr)->y_iters = n_y;
  (*ptr)->x_pitch = x_p;  (*ptr)->y_pitch = y_p;
  (*ptr)->i_call = NULL;
  i_call_ptr = &(*ptr)->i_call;
  }
  else
    {
    if (cifnum < (*ptr)->cifnum)
      Iter_Tree (name, cifnum, n_x, n_y, x_p, y_p, &(*ptr)->l_t_cifnum);
    else if (cifnum == (*ptr)->cifnum)
      Iter_Tree (name, cifnum, n_x, n_y, x_p, y_p, &(*ptr)->e_cifnum);
    else if (cifnum > (*ptr)->cifnum)
      Iter_Tree (name, cifnum, n_x, n_y, x_p, y_p, &(*ptr)->g_t_cifnum);
    }
}
#endif
```

```
#ifneed Mod_Iter
/*************************************************************************
*                                                                        *
*       Module Name:  Mod_Iter                                           *
*       Module Number:  2.3.4                                            *
*       Version/Date:  1.0   17 September 1985                           *
*       Input Parameters:  exit, cifnum, ptr                             *
*       Output Parameters:  exit                                         *
*       Globals Used:  c_port_ptr                                        *
*       Modules Called:  Get_Option, Iter_Parameters, Call_Parameters,   *
*                  C_Port_Names                                          *
*       Calling Modules:  D_Implement                                    *
*       Function:  Modify an iteration of call statements                *
*       PDL:                                                             *
*         If present location is not empty and exit is false            *
*           Walk the call tree to the left;                              *
*           if exit is false and cifnum is valid                         *
*             Print the call statement,                                  *
*             Determine option,                                          *
*             If option is to exit                                       *
*               Set exit to true;                                        *
*             else if option is to modify and mod is to delete           *
*               Delete this call statement;                              *
*             else if option is to modify and mod is to modify           *
*               Get and place new call parameters;                       *
*           If exit is false and there are ports defined                 *
*             Delete or modify ports as needed;                          *
*           If exit is false                                             *
*             Walk the middle call tree;                                 *
*           If exit is false                                             *
*             Walk the right call tree.                                  *
*                                                                        *
*************************************************************************/
Mod_Iter (exit, cifnum, ptr)

int     *exit, cifnum;
struct iter_statement *ptr;
{
char    option, r_x, portname[15];
int     i, exit_ports, finished;

if (ptr != NULL && *exit == FALSE)
  {
  Mod_Iter (exit, cifnum, ptr->l_t_cifnum);
  if (*exit == FALSE && cifnum <= ptr->cifnum)
    {
    exit_ports = FALSE;
    printf ("%s %dx%d ", ptr->cellname, ptr->x_iters, ptr->y_iters);
    printf ("Pitch %d, %d  ", ptr->x_pitch, ptr->y_pitch);
    printf ("T %d, %d  ", ptr->i_call->tran_x, ptr->i_call->tran_y);
    if (ptr->i_call->mir_x)  printf ("Flip UD ");
    if (ptr->i_call->mir_y)  printf ("Flip LR ");
    printf ("R %c", ptr->i_call->rot_x);
    if (ptr->i_call->rot_x == '1') printf ("2");
    printf (" (ret, E, M, or D): ");
    option = Get_Option ();
    if (option == 'E')
      *exit = TRUE;
    else if (option == 'D')
      {
      exit_ports = TRUE;
      ptr->i_call->c_ports = NULL;
      ptr->x_iters = -9999;
      }
    else if (option == 'M')
      {
      Iter_Parameters (&ptr->x_iters, &ptr->y_iters,
                &ptr->x_pitch, &ptr->y_pitch, ptr->i_call->x_min,
                ptr->i_call->y_min, ptr->i_call->x_max, ptr->i_call->y_max);
      Call_Parameters (ptr->cellname, &(ptr->i_call->tran_x),
                &(ptr->i_call->tran_y), &(ptr->i_call->mir_x),
                &(ptr->i_call->mir_y), &(ptr->i_call->rot_x));
    }
    c_port_ptr = ptr->i_call->c_ports;
    if (c_port_ptr != NULL && *exit == FALSE)
    printf ("\nType ret for next port, E to Exit, M to Mod, or D to Delete\n");
    while (*exit == FALSE && exit_ports == FALSE && c_port_ptr != NULL)
      {
      printf ("\n Port %s (ret, E, M, or D): ", c_port_ptr->port_name);
      option = Get_Option ();
      if (option == 'E')
        exit_ports = TRUE;
      else if (option == 'D')
        c_port_ptr->port_name[0] = '-';
      else if (option == 'M')
        {
        printf ("\n New port name : ");
        i = 0;
        while ((c_port_ptr->port_name[i] = getc(0)) != '\n')  ++i;
        c_port_ptr->port_name[i] = '\0';
        }
      c_port_ptr = c_port_ptr->next_name;
      }
    c_port_ptr = &ptr->i_call->c_ports;
    if (*exit == FALSE && exit_ports == FALSE)
      C_Port_Names ();
    }
    if (*exit == FALSE)
      Mod_Iter (exit, cifnum, ptr->e_cifnum);
    if (*exit == FALSE)
      Mod_Iter (exit, cifnum, ptr->g_t_cifnum);
  }
}
#endif
```

```
/**************************************************************************
*                                                                         *
*       Program Name:  MODIFY.C                                           *
*       Version/Date:  1.0  11 August 1985                                *
*                                                                         *
*       Function:  This subprogram to MDCLL.C modifies an already         *
*               existing cell.  Option are to add the cell, delete        *
*               from the cell, or change cell parameters.                 *
*                                                                         *
*       Include Files Needed:  MODLIB.LIB, DPMLIB.LIB,CRE8LIB.LIB,        *
*               STDLIB.C, MDCLL.H (which includes tprintf.c, exec.c,      *
*               and all global variables needed in the mdcll system)      *
*                                                                         *
*       Program Hierarchy:  See next page                                 *
*                                                                         *
**************************************************************************/

#include "mdcll.h"
```

```
/**************************************************************************
*       3.0  MODIFY Main                                                  *
*         |                                                               *
*         |---- 1.1* Name_Cell                                            *
*         |                                                               *
*         |---- 1.2.1* Get_Option                                         *
*         |                                                               *
*         |---- 2.1* Get_Cell                                             *
*         |                                                               *
*         |---- 3.1  Det_Mod                                              *
*         |        |---- 1.2.1* Get_Option                                *
*         |                                                               *
*         |---- 1.2* C_Menu                                               *
*         |                                                               *
*         |---- 1.3* Implement                                            *
*         |                                                               *
*         |---- 3.2  M_Menu                                               *
*         |        |---- 1.2.1* Get_Option                                *
*         |                                                               *
*         |---- 3.3 M_Implement                                           *
*                  |                                                      *
*                  |---- 1.3.6* Print_Work                                *
*                  |                                                      *
*                  |---- 1.3.7* Save_Cell                                 *
*                  |                                                      *
*                  |---- 1.3.1.1* Def_Layer                               *
*                  |                                                      *
*                  |---- 1.3.1.2* Get_Num                                 *
*                  |                                                      *
*                  |---- 3.3.1r Mod_Port                                  *
*                  |         |---- 1.2.1* Get_Option                      *
*                  |         |---- 1.3.1.1* Def_Layer                     *
*                  |         |---- 1.3.1.2* Get_Num                       *
*                  |                                                      *
*                  |---- 3.3.2r Mod_Rect                                  *
*                  |         |---- 1.2.1* Get_Option                      *
*                  |         |---- 1.3.1.2* Get_Num                       *
*                  |         |---- 1.3.2.1* Rect_Tree                     *
*                  |                                                      *
*                  |---- 3.3.3r Mod_Wire                                  *
*                  |         |---- 1.3.6.3.1* Segment_Tree                *
*                  |         |---- 1.2.1* Get_Option                      *
*                  |         |---- 1.3.1.2* Get_Num                       *
*                  |         |---- 1.3.3.1* S_Wire_Tree                   *
*                  |         |---- 1.3.3.3* Place_Wire_Parameters         *
*                  |                                                      *
*                  |---- 3.3.4r Mod_Via                                   *
*                  |         |---- 1.2.1* Get_Option                      *
*                  |         |---- 1.3.1.2* Get_Num                       *
*                  |         |---- 1.3.4.1* Via_Tree                      *
*                  |---- 3.3.5  Mod_Flash                                 *
**************************************************************************/
```

```
/*********************************************************************
*                                                                    *
*       Module Name:  Main                                           *
*       Module Number:  3.0                                          *
*       Version/Date:  1.0   21 August 1985                          *
*       Input Parameters:  none                                      *
*       Output Parameters:  none                                     *
*       Globals Used:  none                                          *
*       Modules Called:  Det_Mod, Name_Cell, Get_Cell, C_Menu,       *
*                       M_Menu, M_Implement, Get_Option, Implement   *
*       Calling Modules:  MDCLL program                              *
*       Function:  Modify an already existing CIF cell               *
*       PDL:                                                         *
*         While not finished modifying                               *
*           Name the cell to be modified;                            *
*           If not found on disk                                     *
*             If shouldn't try search for file again                 *
*               Set finished modifying to true;                      *
*           else                                                     *
*             Set finished modifying to true,                        *
*             Get the cell to be modified,                           *
*             Determine the type of modification,                    *
*             if mod is to add to the cell                           *
*               While create menu option other than quit             *
*                 Implement the desired option,                      *
*             else if mod is to delete or change cell parameters     *
*               While modify option other than quit                  *
*                 Implement the desired modification.                *
*                                                                    *
*********************************************************************/
main()
{
int     finished,
        on_disk;

char    mod,
        option,
        cellname[15],
        filename[12];

printf ("\n\n\n\n\n\n\n\t\t READY TO MODIFY A CELL \n\n\n\n\n\n");
finished = FALSE;
while (finished == FALSE)
  {
  printf ("\n Enter Name of cell to be modified: ");
  Name_Cell (&on_disk, filename, cellname);
```

```
 if (on_disk == FALSE)
  {
  printf ("\n Can't find filename %s; try again (return or N): ", filename);
  if (Get_Option () != '\n')
   finished = TRUE;
  }
 else
  {
  finished = TRUE;
  Get_Cell (filename, cellname);
  mod = Det_Mod ();
  if (mod == 'A')
   while ((option = C_Menu (cellname)) != 'Q')
    Implement (option, filename, cellname);
  else if (mod == 'M')
   while ((option = M_Menu (cellname)) != 'Q')
    M_Implement (option, filename, cellname);
  }
 }
}
#include "modlib.lib"
#include "dpmlib.lib"
#include "cre8lib.lib"
#include "stdlib.c"
```

```
#ifneed Det_Mod
/************************************************************************
*                                                                       *
*       Module Name:  Det_Mod                                           *
*       Module Number:  3.1                                             *
*       Version/Date:  1.0   11 August 1985                             *
*       Input Parameters:  none                                         *
*       Output Parameters:  mod                                         *
*       Globals Used:  none                                             *
*       Modules Called:  Get_Option                                     *
*       Calling Modules:  Main (3.0)                                    *
*       Function: Determine modification: either Add, Modify, or Quit   *
*       PDL:                                                            *
*         While type of modification is not valid                       *
*           Display available options;                                  *
*           Get the option.                                             *
*                                                                       *
************************************************************************/

Det_Mod ()
{
int     mod_valid;
char    mod;

mod_valid = FALSE;
while (mod_valid == FALSE)
  {
  printf ("\n Do you want to \n\n\t\t A = Add to this cell \n");
  printf ("\n\t\t M = Modify cell parameters \n");
  printf ("\n\t\t Q = Quit to operating system \n");
  printf ("\n\n Select Option (A,M,Q): ");
  mod = Get_Option ();
  if (mod == 'A' || mod == 'M' || mod == 'Q')
    mod_valid = TRUE;
    else printf ("\n Option %c not valid \n",mod);
  }
return (mod);
}
#endif
```

```
#ifneed M_Menu
/************************************************************************
*                                                                       *
*       Module Name:  M_Menu                                            *
*       Module Number:  3.2                                             *
*       Version/Date:  1.0   11 August 1985                             *
*       Input Parameters:  cellname                                     *
*       Output Parameters:  none                                        *
*       Globals Used:  none                                             *
*       Modules Called: Get_Option                                      *
*       Calling Modules:  Main (3.0)                                    *
*       Function:  Display options for changes or deletions to a cell   *
*       PDL:                                                            *
*         While option is not valid                                     *
*           Display the menu of options;                                *
*           Get the desired option.                                     *
*                                                                       *
************************************************************************/

M_Menu (cellname)

char    *cellname;
{
char    option;
int     option_valid,
        space_left;

option_valid = FALSE;
while (option_valid == FALSE)
  {
  space_left = AVAIL_SPACE - space_used;
  printf ("\n\t\t\t Memory space used: %d \n", space_left);
  printf ("\n\n Options available for cell %s are: \n\n", cellname);
  printf ("\n Modify a: \t\t\t or: \n\n");
  printf ("\n R = Rectangle (box) \t\t P = Print work \n");
  printf ("\n W = Wire \t\t\t E = Exit to main menu \n");
  printf ("\n V = Via (contact) \t\t     after saving file \n");
  printf ("\n F = round Flash \t\t Q = Quit to system \n");
  printf ("\n I = Interconnection port \n");
  printf ("\n\n SELECT OPTION (R,W,V,F,I,P,E,Q): ");
  option = Get_Option ();
  if (option == 'R' || option == 'W' || option == 'V' || option == 'F'
    ||option == 'I' || option == 'P' || option == 'E' || option == 'Q')
    option_valid = TRUE;
    else printf ("Option %c not valid \n", option);
  }
printf ("\n\n\n\n\n\n\n\n\n\n\n");
return (option);
}
#endif
```

```
#ifneed M_Implement
/*************************************************************************
*                                                                        *
*       Module Name:  M_Implement                                        *
*       Module Number:  3.3                                              *
*       Version/Date:  1.0   11 August 1985                              *
*       Input Parameters:  option, mod, filename, cellname               *
*       Output Parameters:  none                                         *
*       Globals Used:  port_ptr, wire_ptr, f_rect, s_rect, p_rect,       *
*               d_rect, s_via, p_via, d_via                              *
*       Modules Called: Def_Layer, Get_Num, Mod_Rect, Print_Work,        *
*               Mod_Wire, Mod_Via, Mod_Flash, Mod_Port, Save_Cell        *
*       Calling Modules:  Main (3.0)                                     *
*       Function:  Implement functions for changes or deletions          *
*       PDL:                                                             *
*         If option is to print work                                     *
*           Print the work;                                              *
*         else if option is to save the cell                             *
*           Save the cell to disk;                                       *
*         else                                                           *
*           If option other than modify a port                           *
*             Get layer and x location for modification,                 *
*           Print instructions,                                          *
*           If option is to modify an interconnection port               *
*             Modify a port,                                             *
*           else if option is to modify a rectangle                      *
*             Modify a rectangle based on layer,                         *
*           else if option is to modify a wire                           *
*             Modify a wire based on starting layer,                     *
*           else if option is to modify a via                            *
*             Modify a via based on layer,                               *
*           else if option is to modify a flash                          *
*             Modify a flash.                                            *
*                                                                        *
*************************************************************************/
M_Implement (option, filename, cellname)

char    option, *filename, *cellname;

{
int     *exit,
        location;

if (option == 'P')
  Print_Work ();
else if (option == 'E')
  Save_Cell (filename, cellname);
else
  {
    if (option != 'I')
      {
      if (option != 'W')
        {
        printf ("\n Input the layer for this modification\n");
        Def_Layer ();
        }
      printf ("\n Input the x location if known (return or number): ");
      location = Get_Num (0);
      }
    printf ("\n Type return for next, E to Exit, M to Modify, or D to Delete\n");
    *exit = FALSE;
    if (option == 'I')
      Mod_Port (port_ptr);
    else if (option == 'R')
      {
      if (layer == 'F') Mod_Rect (exit, location, f_rect);
      else if (layer == 'S') Mod_Rect (exit, location, s_rect);
      else if (layer == 'D') Mod_Rect (exit, location, d_rect);
      else if (layer == 'P') Mod_Rect (exit, location, p_rect);
      }
    else if (option == 'W')
      Mod_Wire (exit, location, wire_ptr);
    else if (option == 'V')
      {
      if (layer == 'S') Mod_Via (exit, location, s_via);
      else if (layer == 'D') Mod_Via (exit, location, d_via);
      else if (layer == 'P') Mod_Via (exit, location, p_via);
      }
    else if (option == 'F')
      Mod_Flash ();
    }
  }
#endif
```

```
#ifneed Mod_Port
/************************************************************************
*       Module Name:  Mod_Port                                          *
*       Module Number:  3.3.1                                           *
*       Version/Date:  1.0   9 September 1985                           *
*       Input Parameters:  p_ptr                                        *
*       Output Parameters:  none                                        *
*       Globals Used:  layer                                            *
*       Modules Called: Get_Option, Def_Layer, Get_Num                  *
*       Calling Modules:  M_Implement                                   *
*       Function:  Modify the port                                      *
*       PDL:                                                            *
*         If a port is defined at this location                         *
*           If port hasn't been deleted                                 *
*             Print port parameters;                                    *
*             Determine option;                                         *
*             if option is to delete, Delete the port;                  *
*             else if option is to modify                               *
*               Get new port parameters;                                *
*           if not exit                                                 *
*             Go to next port.                                          *
************************************************************************/
Mod_Port (p_ptr)
struct inter_ports *p_ptr;
{
char    option;
int     i;
if (p_ptr != NULL)
  {
  if (p_ptr->y_loc != -9999)
    {
    printf ("\n Port %s %c ", p_ptr->port_name, p_ptr->port_layer);
    printf ("%d,%d (ret, E, M, or D): ", p_ptr->x_loc, p_ptr->y_loc);
    option = Get_Option ();
    if (option == 'D')
      p_ptr->y_loc = -9999;
    else if (option == 'M')
      {
      printf ("\n New Port Name: ");
      i = 0;
      while ((p_ptr->port_name [i] = getc(0)) != '\n') ++i;
      p_ptr->port_name [i] = '\0';
      printf ("\n New port layer \n");  Def_Layer (FALSE);
      p_ptr->port_layer = layer;
      printf ("\n New X Location : ");  p_ptr->x_loc = Get_Num (0);
      printf ("\n New Y Location : ");  p_ptr->y_loc = Get_Num (0);
      }
    if (option != 'E')  Mod_Port (p_ptr->next_port);
    }
  }
}
#endif
```

```
#ifneed Mod_Rect
/************************************************************************
*                                                                       *
*       Module Name:  Mod_Rect                                          *
*       Module Number:  3.3.2                                           *
*       Version/Date:  1.0   11 August 1985                             *
*       Input Parameters:  exit, mod, location, l_rect                  *
*       Output Parameters:  exit                                        *
*       Globals Used:  f_rect, s_rect, p_rect, d_rect                   *
*       Modules Called: Get_Option, Get_Num, Rect_Tree                  *
*       Calling Modules:  M_Implement                                   *
*       Function:  Modify an existing rectangle                         *
*       PDL:                                                            *
*         If a rectangle at this location and exit flag is false        *
*           Walk the left rectangle tree;                               *
*           If not exit and location valid                              *
*             Print the rectangle parameters,                           *
*             Determine option for this rectangle,                      *
*             If option is to exit                                      *
*               Set exit flag to true;                                  *
*             else if option is to modify                               *
*               Get new rectangle parameters,                           *
*               If different x location                                 *
*                 Delete this rectangle,                                *
*                 Place new rectangle in rect tree;                     *
*             else if option is to delete                               *
*               Delete this rectangle;                                  *
*           If not exit                                                 *
*             Walk the middle rect tree;                                *
*           If not exit                                                 *
*             Walk the right rect tree.                                 *
*                                                                       *
************************************************************************/

Mod_Rect (exit, location, l_rect)

int     *exit, location;
struct rectangle  *l_rect;

{
char    option;
int     r_h, r_l, r_x, r_y;

if (l_rect != NULL && *exit == FALSE)
  {
  Mod_Rect (exit,location,(l_rect)->less_than_x);
  if (*exit == FALSE && (location <= l_rect->x_location || location == 0))
    {
    printf ("\nRect H = %d, L = %d, ",l_rect->height, l_rect->length);
    printf ("x,y = %d, %d ", l_rect->x_location, l_rect->y_location);
    printf (" (return, E, M, or D): ");
    option = Get_Option ();
    if (option == 'E')
      *exit = TRUE;
    else if (option == 'M')
      {
      printf ("\n New Height: ");
      r_h = (l_rect)->height = Get_Num (0);
      printf ("\n New Length: ");
      r_l = (l_rect)->length = Get_Num (0);
      printf ("\n New x location: ");
      r_x = Get_Num (0);
      printf ("\n New y location: ");
      r_y = l_rect->y_location = Get_Num (0);
      if (r_x != l_rect->x_location)
        {
        l_rect->height = l_rect->length = 0;
        if (layer == 'F') Rect_Tree (r_h, r_l, r_x, r_y, &f_rect);
        else if (layer == 'S') Rect_Tree (r_h, r_l, r_x, r_y, &s_rect);
        else if (layer == 'D') Rect_Tree (r_h, r_l, r_x, r_y, &d_rect);
        else if (layer == 'P') Rect_Tree (r_h, r_l, r_x, r_y, &p_rect);
        }
      }
    else if (option == 'D')
      l_rect->height = l_rect->length = 0;
    }
  if (*exit == FALSE)
    Mod_Rect (exit, location, l_rect->equals_x);
  if (*exit == FALSE)
    Mod_Rect (exit, location, l_rect->greater_than_x);
  }
}
#endif
```

```
#ifneed Mod_Wire
/************************************************************************
*       Module Name:  Mod_Wire                                          *
*       Module Number:  3.3.3                                           *
*       Version/Date:  1.0   11 August 1985                             *
*       Input Parameters:  exit, mod, location, ptr                     *
*       Output Parameters:  exit                                        *
*       Globals Used:  seg_ptr, wire_ptr, layer                         *
*       Modules Called:  Segment_Tree, Get_Option, Get_Num,             *
*                       S_Wire_Tree, Place_Wire_Parameters              *
*       Calling Modules:  M_Implement                                   *
*       Function:  Modify an existing wire                              *
*       PDL:                                                            *
*         If a wire is defined at this location and exit flag is false  *
*           Walk the left wire tree;                                    *
*           if exit false and location valid and wire not deleted       *
*             Print the wire parameters,                                *
*             Determine option for this wire,                           *
*             if option is to exit, Set exit flag to true;              *
*             else if option is to modify                               *
*               Get new starting x,y location,                          *
*               If the new x = the old x location                       *
*                 Display and change segment parameters as needed;      *
*               else                                                    *
*                 Place new wire using new x loc with old segments,     *
*                 Delete old wire;                                      *
*             else if option is to delete                               *
*               Delete this wire's segments;                            *
*           if not exit, Walk the middle wire tree;                     *
*           if not exit, Walk the right wire tree.                      *
************************************************************************/
Mod_Wire (exit, location, ptr)
int    *exit, location;
struct wire  *ptr;
{
char    option, temp_layer;
int     new_x, exit_seg;
struct wire_parameters *seg;
if (ptr != NULL && *exit == FALSE)
  {
  Mod_Wire (exit,location,ptr->w_l_t_x);
  if (*exit == FALSE && (location <= ptr->w_x_location || location == 0)
          && ptr->w_param != NULL)
    {
    printf ("\nWire x,y = %d, %d", ptr->w_x_location, ptr->w_y_location);
    Segment_Tree (0, ' ', ptr->w_param);
    printf ("\n\t (return, E, M, or D): ");
    option = Get_Option ();
    if (option == 'E')
      *exit = TRUE;
    else if (option == 'M')
      {
```

```
   printf ("\n New x location: ");
   new_x = Get_Num (0);
   printf ("\n New y location: ");
   ptr->w_y_location = Get_Num (0);
   if (new_x == ptr->w_x_location)
    {
    seg = ptr->w_param;
    exit_seg = FALSE;
    while (seg != NULL && exit_seg == FALSE)
     {
     printf ("%c W%d %c", seg->w_layer,seg->width,seg->direction);
     printf ("%d (return, E, or M): ", seg->length);
     option = Get_Option ();
     if (option == 'E')
      exit_seg = TRUE;
      else if (option == 'M')
       {
       printf ("\nNew width : ");
       seg->width = Get_Num (0);
       printf ("\nNew length : ");
       seg->length = Get_Num (0);
       printf ("\nNew Layer");
       Def_Layer(FALSE);
       seg->w_layer= layer;
       }
     seg = seg->next_segment;
     }
    }
    else
    {
    S_Wire_Tree (new_x, ptr->w_y_location, &wire_ptr);
    seg = ptr->w_param;
    ptr->w_param = NULL;
    temp_layer = layer;
    while (seg != NULL)
     {
     layer = seg->w_layer;
     Place_Wire_Parametr(seg->width,seg->length,seg->direction,seg_ptr);
     seg = seg->next_segment;
     }
    layer = temp_layer;
    }
   }
  else if (option == 'D') ptr->w_param = NULL;
  }
 if (*exit == FALSE)
  Mod_Wire (exit, location, ptr->w_e_x);
 if (*exit == FALSE)
  Mod_Wire (exit, location, ptr->w_g_t_x);
 }
}
#endif
```

```
#ifneed Mod_Via
/************************************************************************
*                                                                       *
*       Module Name:  Mod_Via                                           *
*       Module Number:  3.3.4                                           *
*       Version/Date:  1.0   11 August 1985                             *
*       Input Parameters:  exit, mod, location, l_via                   *
*       Output Parameters:  none                                        *
*       Globals Used:  s_via, p_via, d_via                              *
*       Modules Called:  Get_Option, Get_Num, Via_Tree                  *
*       Calling Modules:  M_Implement                                   *
*       Function:  Modify an existing Via                               *
*       PDL:                                                            *
*         If a via is defined at this location and exit flag is false  *
*           Walk the left via tree;                                     *
*           If not exit and location valid and via not deleted          *
*             Print via location,                                       *
*             Determine option,                                         *
*             if option is to exit                                      *
*               Set exit flag to true;                                  *
*             else if option is to modify                               *
*               Get new x,y location,                                   *
*               If new x does not = old x                               *
*                 Place a via with new x,y,                             *
*                 Delete old via;                                       *
*             else if option is to delete                               *
*               Delete via;                                             *
*           If not exit                                                 *
*             Walk middle via tree;                                     *
*           If not exit                                                 *
*             Walk right via tree.                                      *
*                                                                       *
************************************************************************/

Mod_Via (exit, location, l_via)

int     *exit, location;
struct via  *l_via;

{
char    option;
int     new_x;

if (l_via != NULL && *exit == FALSE)
  {
  Mod_Via (exit, location, l_via->v_l_t_x);
  if (*exit == FALSE && location <= l_via->v_x_location
         && l_via->v_y_location != -9999)
    {
    printf ("\nVia x,y = %d, %d ", l_via->v_x_location, l_via->v_y_location);
    printf (" (return, E, M, or D): ");
    option = Get_Option ();
```

```
        if (option == 'E')
          *exit = TRUE;
        else if (option == 'M')
          {
          printf ("\n New x location: ");
          new_x = Get_Num (0);
          printf ("\n New y location: ");
          l_via->v_y_location = Get_Num (0);
          if (new_x != l_via->v_x_location)
            {
            if (layer == 'S') Via_Tree (new_x, l_via->v_y_location, &s_via);
            else if (layer == 'D') Via_Tree (new_x, l_via->v_y_location, &d_via);
            else if (layer == 'P') Via_Tree (new_x, l_via->v_y_location, &p_via);
            l_via->v_y_location = -9999;
            }
          }
        else if (option == 'D')
          l_via->v_y_location = -9999;
        }
      if (*exit == FALSE)
        Mod_Via (exit, location, l_via->v_e_x);
      if (*exit == FALSE)
        Mod_Via (exit, location, l_via->v_g_t_x);
      }
    }
    #endif
```

```
#ifneed Mod_Flash
/************************************************************************
*                                                                       *
*       Module Name:  Mod_Flash                                         *
*       Module Number:  3.3.5                                           *
*       Version/Date:  1.0   11 August 1985                             *
*       Input Parameters:  none                                         *
*       Output Parameters:  none                                        *
*       Globals Used:  none                                             *
*       Modules Called:  none                                           *
*       Calling Modules:  M_Implement                                   *
*       Function:  Modify round flashes                                 *
*       PDL:                                                            *
*                                                                       *
************************************************************************/
Mod_Flash ()

{
printf ("\n Mod_Flash not yet coded \n");
}

#endif
```

```
/************************************************************************
*                                                                       *
*       Program Name:  INTER.C                                          *
*       Version/Date:  1.0  18 September 1985                           *
*                                                                       *
*       Function:  This subprogram to MDCLL.C does nothing right now    *
*               but will eventually allow the user to interconnect      *
*               already existing cells.                                 *
*                                                                       *
*       Include Files Needed:  MDCLL.H (which includes tprintf.c,      *
*               exec.c, and all global variables needed in the          *
*               mdcll system)                                           *
*                                                                       *
*       Program Hierarchy:                                              *
*                                                                       *
*       4.0 INTER Main                                                  *
*         |                                                             *
*         |---- 0.0* MDCLL Main                                         *
*                                                                       *
************************************************************************/

#include "mdcll.h"
```

```
/************************************************************************
*                                                                       *
*       Module Name:  Main                                              *
*       Module Number:  4.0                                             *
*       Version/Date:  1.0   18 September 1985                          *
*       Input Parameters:  none                                         *
*       Output Parameters:  none                                        *
*       Globals Used:  none                                             *
*       Modules Called:  none                                           *
*       Calling Modules:  MDCLL program                                 *
*       Function:  Eventually interconnect already placed cells         *
*       PDL:                                                            *
*         Print message;                                                *
*         Return to MDCLL.                                              *
*                                                                       *
************************************************************************/

main()
{
printf ("\n\n INTER Subprogram not yet implemented\n\n\n");
printf ("\n\n To interconnect cells use the modify subprogram ");
printf ("\n to place wire statements, vias, etc.\n");
printf ("\n\n Hit return to continue: ");
while (getc(0) != '\n');
exec ("MDCLL", "N");
}
```

```
/************************************************************************
*                                                                       *
*       Program Name:  PRIPLO.C                                         *
*       Version/Date:  1.0  18 September 1985                           *
*                                                                       *
*       Function:  This subprogram to MDCLL.C prints an already         *
*               existing cell.                                          *
*                                                                       *
*       Include Files Needed:  MODLIB.LIB, DPMLIB.LIB,CRE8LIB.LIB,      *
*               STDLIB.C, MDCLL.H (which includes tprintf.c, exec.c,    *
*               and all global variables needed in the mdcll system)    *
*                                                                       *
*       Program Hierarchy:                                              *
*                                                                       *
*       5.0  PRIPLO Main                                                *
*         |                                                             *
*         |---- 1.1* Name_Cell                                          *
*         |                                                             *
*         |---- 1.2.1* Get_Option                                       *
*         |                                                             *
*         |---- 2.1.1* In_File                                          *
*         |                                                             *
*         |---- 0.0* MDCLL Main                                         *
*                                                                       *
************************************************************************/

#include "mdcll.h"
```

```
/*************************************************************************
*                                                                        *
*       Module Name:  Main                                               *
*       Module Number:  5.0                                              *
*       Version/Date:  1.0   18 September 1985                           *
*       Input Parameters:  none                                          *
*       Output Parameters:  none                                         *
*       Globals Used:  none                                              *
*       Modules Called:  Name_Cell, In_File, Get_Option                  *
*       Calling Modules:  MDCLL program                                  *
*       Function:  Print an already existing CIF cell                    *
*       PDL:                                                             *
*         While not finished                                             *
*           Name the cell to be printed;                                 *
*           If not found on disk                                         *
*             If shouldn't try search for file again                     *
*               Set finished to true;                                    *
*           else                                                         *
*             Set finished to true,                                      *
*             Send the file to the terminal one line at a time.          *
*                                                                        *
*************************************************************************/
main()
{
int     finished,
        on_disk,
        i;

char    option,
        cellname[15],
        filename[12],
        string[80];

FILE    *fopen(), *fp;

printf ("\n\n\n\n\n\n\n\t\t READY TO PRINT A CELL \n\n\n\n\n\n");
finished = FALSE;
while (finished == FALSE)
  {
  printf ("\n Enter Name of cell to be printed: ");
  Name_Cell (&on_disk, filename, cellname);
  if (on_disk == FALSE)
    {
    printf ("\n Can't find filename %s; try again (return or N): ", filename);
    if (Get_Option () != '\n')
      finished = TRUE;
    }
  else
    {
    finished = TRUE;
    fp = fopen (filename, "r");
    while (string[0] != 'E' && option != 'E')
      {
      In_File (fp, '\n', string);
      printf ("%s \n",string);
      if (i > 19 || string [0] == 'E')
        {
        printf ("\n\n Hit Return to Continue or E to exit: ");
        option = Get_Option ();
        if (option != '\n')
          option = 'E';
        i = 1;
        }
      else ++i;
      }
    fclose (fp);
    }
  }
exec ("MDCLL", "N");
}
#include "dpmlib.lib"
#include "cre8lib.lib"
#include "savelib.lib"
#include "stdlib.c"
```

```
/*************************************************************************
*                                                                        *
*       Program Name:  FINALIZE.C                                        *
*       Version/Date:  1.0  5 September 1985                             *
*                                                                        *
*       Function:  This subprogram to MDCLL.C is used to manipulate      *
*               already existing basic cells to arrange them into        *
*               a single CIF file.                                       *
*                                                                        *
*       Library Files Needed:  MDCLL.H (which includes tprintf.c         *
*               exec.c, and all global variables needed in the           *
*               MDCLL system), CLIB.C, MLIB.C, SAVELIB.C  and STDLIB.C.  *
*                                                                        *
*       Program Hierarchy:                                               *
*                                                                        *
*       6.0  FINALIZE Main                                               *
*         |                                                              *
*         |---- 1.1* Name_Cell                                           *
*         |                                                              *
*         |---- 2.1.1* In_File                                           *
*         |                                                              *
*         |---- 2.1.6* Get_Call                                          *
*         |                                                              *
*         |---- 2.1.7* Get_Iter                                          *
*         |                                                              *
*         |---- 6.1  Format_Cif                                          *
*         |       |                                                      *
*         |       |---- 6.1.1r Primary_Calls                             *
*         |       |          |---- 6.1.1.1r Call_Names                   *
*         |       |                                                      *
*         |       |---- 6.1.2r Iter_Calls                                *
*         |       |          |---- 6.1.1.1* Call_Names                   *
*         |       |                                                      *
*         |       |---- 6.1.3  Process_Call                              *
*         |       |          |---- 1.2.1* Get_Option                     *
*         |       |          |---- 2.1.1* In_File                        *
*         |       |          |---- 7.2.1* Out_File                       *
*         |       |          |---- 6.1.1.1* Call_Names                   *
*         |       |          |---- 7.1* Det_Cifnum                       *
*         |       |                                                      *
*         |       |---- 2.1.1* In_File                                   *
*         |       |                                                      *
*         |       |---- 7.2.1* Out_File                                  *
*         |                                                              *
*         |---- 1.2.1* Get_Option                                        *
*         |                                                              *
*         |---- 0.0* MDCLL Main                                          *
*                                                                        *
*************************************************************************/

#include "mdcll.h"
```

```
/************************************************************************
*                                                                       *
*       Module Name:  Main                                              *
*       Module Number:  6.0                                             *
*       Version/Date:  1.0   5 September 1985                           *
*       Input Parameters:  none                                         *
*       Output Parameters:  none                                        *
*       Globals Used:  none                                             *
*       Modules Called: Name_Cell, Get_Option, Get_Call, Format_Cif,    *
*               In_File, Get_Iter, MDCLL Main                           *
*       Calling Modules:  MDCLL program                                 *
*       Function:  Format all needed CIF for one circuit into one file  *
*       PDL:                                                            *
*         While filename not found                                      *
*           Name the cell to be finalized;                              *
*           If file found on disk                                       *
*             Open the file,                                            *
*             While not finished reading the file                       *
*               Get all the cell names,                                 *
*             Close the file,                                           *
*             Format the final CIF;                                     *
*           else                                                        *
*             If don't want to search disk for another filename         *
*               Say the filename was found to exit loop;                *
*         Execute MDCLL.                                                *
*                                                                       *
************************************************************************/
main ()

{
int     on_disk,
        finished_reading;

char    filename [12],
        cellname [15],
        string [80];

FILE    *fopen(), *fp;

printf ("\n\n\n\n\n\n\n\t\t  READY TO FINALIZE A CELL \n\n\n\n\n\n\n");
on_disk = FALSE;
while (on_disk == FALSE)
  {
  printf ("\n Enter name of cell to be finalized: ");
  Name_Cell (&on_disk, filename, cellname);
  if (on_disk == TRUE)
    {
    fp = fopen (filename, "r");
    finished_reading = FALSE;
  while (finished_reading == FALSE)
   {
   In_File (fp, '\n', string);
   if (string [0] == '(' && string [1] == 'S' && string [2] == 'T'
        && string [3] == 'A' )
    {
    if (string [7] == 'C')
     Get_Call (fp, string);
    else if (string [7] == 'I')
     Get_Iter (fp, string);
    }
   else if (string [0] == 'E')
    finished_reading = TRUE;
   }
  fclose (fp);
  Format_Cif (filename, cellname);
  }
  else
  {
  printf ("\n Can't find file %s; Try again (ret or N): ");
  if (Get_Option () != '\n')
   on_disk = TRUE;
  }
 }
exec ("MDCLL", "N");
}


#include "finallib.lib"
#include "dpmlib.lib"
#include "modlib.lib"
#include "cre8lib.lib"
#include "savelib.lib"
#include "stdlib.c"
```

```
/***********************************************************************
*                                                                      *
*       Module Name:  Format_Cif                                       *
*       Module Number:  6.1                                            *
*       Version/Date:  1.0   5 September 1985                          *
*       Input Parameters:  filename, cellname                          *
*       Output Parameters:  none                                       *
*       Globals Used:  call_ptr, iter_ptr                              *
*       Modules Called: Primary_Calls, Process_Call, Iter_Calls        *
*       Calling Modules:  FINALIZE Main (6.0)                          *
*       Function:  Format CIF                                          *
*       PDL:                                                           *
*         Open the file final.cif;                                     *
*         Load the primary calls into the secondary call tree;         *
*         While there are more secondary calls                         *
*           Process the calls;                                         *
*         Reopen the file being finalized;                             *
*         Copy the file to final.cif one line at a time;               *
*         Close both files.                                            *
*                                                                      *
***********************************************************************/
Format_Cif (filename, cellname)

char    *filename, *cellname;

{
int     finished;
char    a_cifnum [7], string [80];
FILE    *fopen(), *fp1, fp2;
struct called_cells *scptr;

fp1 = fopen ("final.cif", "w");
Primary_Calls (call_ptr);
Iter_Calls (iter_ptr);
scptr = s_calls;
while (scptr != NULL)
  {
  Process_Call (scptr->cellname, fp1);
  scptr = scptr->next_call;
  }
fp2 = fopen (filename, "r");
finished = FALSE;
while (finished == FALSE)
  {
  In_File (fp2, '\n', string);
  Out_File (string, "\n", fp1);
  if (string [0] == 'E')   finished = TRUE;
  }
fclose (fp2);
fclose (fp1);
}
```

```
/************************************************************************
*                                                                       *
*       Module Name:  Primary_Calls                                     *
*       Module Number:  6.1.1                                           *
*       Version/Date:  1.0   5 September 1985                           *
*       Input Parameters:  cptr                                         *
*       Output Parameters:  none                                        *
*       Globals Used:  s_calls                                          *
*       Modules Called: Call_Names, Primary_Calls (recursion)           *
*       Calling Modules:  Format_Cif, Primary_Calls                     *
*       Function:  Get all the called cells names.                      *
*       PDL:                                                            *
*         If the location is defined                                    *
*           Walk the left call tree;                                    *
*           Extract the called cells' names and cifnums;                *
*           Walk the right call tree.                                   *
*                                                                       *
************************************************************************/
Primary_Calls (cptr)

struct call_statement *cptr;

{
if (cptr != NULL)
  {
  Primary_Calls (cptr->l_t_name);
  Call_Names (cptr->cellname, cptr->cifnum, &s_calls);
  Primary_Calls (cptr->g_t_name);
  }
}
```

```
/************************************************************************
*                                                                       *
*       Module Name:  Call_Names                                        *
*       Module Number:  6.1.1.1                                         *
*       Version/Date:  1.0   5 September 1985                           *
*       Input Parameters:  cellname, cifnum, scptr                      *
*       Output Parameters:  none                                        *
*       Globals Used:  none                                             *
*       Modules Called: Call_Names (recursion)                          *
*       Calling Modules:  Call_Names, Primary_Calls, Process_Call       *
*       Function:  Places all calls not previously defined in the       *
*                 called cells tree                                     *
*       PDL:                                                            *
*         If location is not defined (call not a duplicate)             *
*           Allocate memory,                                            *
*           Store the cellname and cifnum;                              *
*         else                                                          *
*           If the called cell's name is not already stored             *
*             Check the next call.                                      *
*                                                                       *
************************************************************************/
Call_Names (cellname, cifnum, scptr)

char    *cellname;
int     cifnum;
struct  called_cells  **scptr;

{
int     i;

if (*scptr == NULL)
  {
  *scptr = alloc (sizeof (struct called_cells));
  (*scptr)->next_call = NULL;
  i = 0;
  while (((*scptr)->cellname[i] = cellname[i]) != '\0') ++i;
  (*scptr)->cifnum = cifnum;
  }
  else
  if (cifnum != (*scptr)->cifnum)
    Call_Names (cellname, cifnum, &(*scptr)->next_call);
}
```

```
/************************************************************************
*                                                                       *
*       Module Name:  Iter_Calls                                        *
*       Module Number:  6.1.2                                           *
*       Version/Date:  1.0   18 September 1985                          *
*       Input Parameters:  iptr                                         *
*       Output Parameters:  none                                        *
*       Globals Used:  s_calls                                          *
*       Modules Called: Call_Names, Iter_Calls (recursion)              *
*       Calling Modules:  Format_Cif, Iter_Calls                        *
*       Function:  Get all the iterated cells names.                    *
*       PDL:                                                            *
*         If the location is defined                                    *
*           Walk the left iter tree;                                    *
*           Extract the iterated cells' names and cifnums;              *
*           Walk the right iter tree.                                   *
*                                                                       *
************************************************************************/

Iter_Calls (iptr)

struct iter_statement *iptr;

{
if (iptr != NULL)
  {
  Iter_Calls (iptr->l_t_cifnum);
  Call_Names (iptr->cellname, iptr->cifnum, &s_calls);
  Iter_Calls (iptr->g_t_cifnun);
  }
}
```

```
/***********************************************************************
*                                                                      *
*       Module Name:  Process_Call                                     *
*       Module Number:  6.1.3                                          *
*       Version/Date:  1.0   5 September 1985                          *
*       Input Parameters:  cellname, fp1                               *
*       Output Parameters:  none                                       *
*       Globals Used:  s_calls                                         *
*       Modules Called:  Get_Option, In_File, Out_File, Call_Names,    *
*                        Det_Cifnum                                    *
*       Calling Modules:  Format_Cif                                   *
*       Function:  Copy called cells file an ensure no files are missed *
*       PDL:                                                           *
*         Obtain the filename from the cellname;                       *
*         While not finished                                           *
*           Open the file,                                             *
*           Copy a line to final.cif,                                  *
*           If the line is a call statement                            *
*             Add the call to the called cells tree.                   *
*                                                                      *
***********************************************************************/
Process_Call (cellname, fp1)

char    *cellname;
FILE    *fp1;

{
int     i, j, finished;
char    filename [20], string [80];
FILE    *fopen (), *fp2;

i = 0;
while ((filename [i] = cellname [i]) != '\0') ++i;
filename [i++] = '.';
filename [i++] = 'C';
filename [i++] = 'I';
filename [i++] = 'F';
filename [i] = '\0';
finished = FALSE;
while (finished == FALSE)
  {
  fp2 = fopen (filename, "r");
  if (fp2 == NULL)
    {
    printf ("\n Can't find file %s; Try again (ret or N): ", filename);
    if (Get_Option () != '\n') finished = TRUE;
    }
    else
    {
```

```
    while (finished == FALSE)
      {
      In_File (fp2, '\n', string);
      printf ("%s\n", string);
      Out_File (string, "\n", fp1);
      if (string[0] == 'D' && string[1] == 'F' && string[2] == ';')
        {
        Out_File ("\n\n", "\n\n", fp1);
        finished = TRUE;
        }
      else if (string [0] == '(' && (string [1] == 'C' || string [1] == 'I'))
        {
        i = 6;  j = 0;
        while ((cellname [j++] = string [i++]) != ' ');
        cellname [j] = '\0';
        Call_Names (cellname, Det_Cifnum (cellname), &s_calls);
        }
      }
    fclose (fp2);
    }
  }
}
```

```
/*************************************************************************
*       Program Name:  SAVECELL                                          *
*       Version/Date:  1.0  29 August 1985                               *
*       Function:  This subprogram in the MDCLL system is used to        *
*               save the cells to disk.                                  *
*       Library Files Needed:  MDCLL.H, SAVELIB.C, STDLIB.C              *
*       Program Hierarchy:                                               *
*       7.0  SAVECELL Main                                               *
*       |                                                                *
*       |---- 7.1  Det_Cifnum                                            *
*       |                                                                *
*       |---- 7.2  Header_Out_File                                       *
*       |       |---- 7.2.1 Out_File                                     *
*       |                                                                *
*       |---- 7.3  Out_Cell_Layout                                       *
*       |       |                                                        *
*       |       |---- 7.2.1* Out_File                                    *
*       |       |                                                        *
*       |       |---- 7.3.1r Out_Port_Tree                               *
*       |       |         |---- 7.2.1* Out_File                          *
*       |       |                                                        *
*       |       |---- 7.3.2r Out_Rect_Tree                               *
*       |       |         |---- 7.2.1* Out_File                          *
*       |       |                                                        *
*       |       |---- 7.3.3r Out_Wire_Tree                               *
*       |       |         |---- 7.2.1* Out_File                          *
*       |       |         |---- 7.3.3.1r Out_W_Comment                   *
*       |       |         |           |---- 7.2.1* Out_File              *
*       |       |         |---- 7.3.3.2r Out_W_Segment                   *
*       |       |                     |---- 7.2.1* Out_File              *
*       |       |                     |---- 1.3.2.1* Rect_Tree           *
*       |       |                     |---- 1.3.4.1* Via_Tree            *
*       |       |                                                        *
*       |       |---- 7.3.4r Out_Via_Tree                                *
*       |       |         |---- 7.2.1* Out_File                          *
*       |       |                                                        *
*       |       |---- 7.3.5r Out_Call_Tree                               *
*       |       |         |---- 7.2.1* Out_File                          *
*       |       |         |---- 7.3.5.1 Out_C_Statement                  *
*       |       |         |           |---- 7.2.1* Out_File              *
*       |       |         |---- 7.3.5.2 Out_P_Names                      *
*       |       |                     |---- 7.2.1* Out_File              *
*       |       |                                                        *
*       |       |---- 7.3.6r Out_Iter_Tree                               *
*       |                 |---- 7.2.1* Out_File                          *
*       |                 |---- 7.3.5.1* Out_C_Statement                 *
*       |                 |---- 7.3.5.2* Out_P_Names                     *
*       |                                                                *
*       |---- 7.4  Footer_Out_File                                       *
*       |       |---- 7.2.1* Out_File                                    *
*       |---- 0.0* MDCLL Main                                            *
*************************************************************************/
```

```
#include "mdcll.h"
/************************************************************************
*       Module Name:  Main                                              *
*       Module Number:  7.0                                             *
*       Version/Date:  1.0  29 August 1985                              *
*       Input Parameters:  argument list containing filename,           *
*               cellname, and all the pointers for the cell to be saved *
*       Output Parameters:  none                                        *
*       Globals Used:  port_ptr, f_rect, s_rect, p_rect, d_rect, s_via  *
*               p_via, d_via, wire_ptr, call_ptr, iter_ptr              *
*       Modules Called:  Det_Cifnum, Header_Out_File, Out_Cell_Layout   *
*                       Footer_Out_File, MDCLL Main                     *
*       Function:  Save the cif file to disk                            *
*       PDL:                                                            *
*         Convert the pointers for the cell to be saved;                *
*         Determine the cifnum;                                         *
*         Open the file;                                                *
*         Output the CIF file header;                                   *
*         Output the CIF cell layout;                                   *
*         Output the CIF file footer;                                   *
*         Close the file;                                               *
*         Execute MDCLL.                                                *
************************************************************************/
main (argc, argv)
int     argc;
char    *argv[];
{
char    filename[19], cellname[15], a_cifnum[7];
int     cifnum, i, j;
FILE    *fopen(), *fp;
i = 0;  j = 1;
while ((filename[i] = argv[j][i]) != '\0') ++i;
i = 0;  ++j;
while ((cellname[i] = argv[j][i]) != '\0') ++i;
port_ptr = atoi (argv [++j]);
f_rect = atoi (argv[++j]);  s_rect = atoi (argv[++j]);
d_rect = atoi (argv[++j]);  p_rect = atoi (argv[++j]);
s_via = atoi (argv[++j]);  d_via = atoi (argv[++j]);
p_via = atoi (argv[++j]);  wire_ptr = atoi (argv[++j]);
call_ptr = atoi (argv[++j]);  iter_ptr = atoi (argv[++j]);
cifnum = Det_Cifnum (cellname);
itoa (cifnum, a_cifnum);
fp = fopen(filename, "w");
Header_Out_File (cellname, a_cifnum, fp);
Out_Cell_Layout (fp);
Footer_Out_File (cellname, a_cifnum, fp);
fclose (fp);
exec ("mdcll", "N");
}
#include "savelib.lib"
#include "cre8lib.lib"
#include "stdlib.c"
```

```
#ifneed Det_Cifnum
/*************************************************************************
*                                                                        *
*       Module Name:  Det_Cifnum                                         *
*       Module Number:  7.1                                              *
*       Version/Date:  1.0  25 July 1985                                 *
*       Input Parameters:  cellname                                      *
*       Output Parameters:  cifnum                                       *
*       Globals Used:  none                                              *
*       Modules Called:  none                                            *
*       Calling Modules:  SAVECELL Main, D_Implement, Process_Call       *
*       Function:  Determine the CIF number based on the cellname        *
*       PDL:                                                             *
*         Compute cifnum from characters and their position;            *
*         If calculated cifnum is negative                               *
*           Make it positive;                                            *
*         Ensure the cifnum is greater than 1000.                        *
*         Return the cifnum.                                             *
*                                                                        *
*************************************************************************/

Det_Cifnum (cellname)
char    *cellname;
{
int     i,
        cifnum;

i = 0;
cifnum = 0;
while ((cellname[i] != '\0') && (cellname[i] != '\n'))
  {
  cifnum = 10 * cifnum + cellname[i] - '0';
  ++i;
  }
if (cifnum < 0)
  cifnum = cifnum * -1;
cifnum = cifnum + 1000;
return (cifnum);
}

#endif
```

```
#ifneed Header_Out_File
/*************************************************************************
*                                                                        *
*       Module Name:  Header_Out_File                                    *
*       Module Number:  7.2                                              *
*       Version/Date:  1.0   28 July 1985                                *
*       Input Parameters:  cellname, a_cifnum, fp                        *
*       Output Parameters:  none                                         *
*       Globals Used:  SCALE_A, SCALE_B                                  *
*       Modules Called:  Out_File                                        *
*       Calling Modules:  Main (7.0)                                     *
*       Function:  Print the CIF file header                             *
*       PDL:  N/A                                                        *
*                                                                        *
*************************************************************************/

Header_Out_File (cellname, a_cifnum, fp)
FILE    *fp;

{
printf ("Initializing cell %s\n", cellname);
Out_File ("DS", " ", fp);
Out_File (a_cifnum, " ", fp);
Out_File (SCALE_A, " ", fp);
Out_File (SCALE_B, ";\n", fp);
Out_File ("(TITLE", " ", fp);
Out_File (cellname, " ", fp);
Out_File (a_cifnum, ");\n", fp);
}

#endif
```

```
#ifneed Out_File
/************************************************************************
*                                                                       *
*       Module Name:  Out_File                                          *
*       Module Number:  7.2.1                                           *
*       Version/Date:  1.0   27 July 1985                               *
*       Input Parameters:  string, separator, fp                        *
*       Output Parameters:  none                                        *
*       Globals Used:  none                                             *
*       Modules Called:  none                                           *
*       Calling Modules:  Header_Out_File, Out_Cell_Layout,             *
*                       Out_Port_Tree, Out_Walk_Tree, Out_Call_Tree     *
*                       Out_File_Footer, Process_Call, Format_Cif,      *
*                       Out_C_Statement, Out_P_Names                    *
*       Function:  Output strings to the disk                           *
*       PDL:  N/A                                                       *
*                                                                       *
************************************************************************/

Out_File (string, separator, fp)
register char   *string,
                *separator;
register FILE *fp;

{
register int c;

while (c = *string++)
  putc (c, fp);
while (c = *separator++)
  putc (c, fp);
}

#endif
```

```
#ifneed Out_Cell_Layout
/*************************************************************************
*       Module Name:  Out_Cell_Layout                                    *
*       Module Number:  7.3                                              *
*       Version/Date:  1.0   28 July 1985                                *
*       Input Parameters:  fp                                            *
*       Output Parameters:  none                                         *
*       Globals Used:  port_ptr, f_rect, s_rect, p_rect, d_rect, s_via,  *
*               p_via, d_via, wire_ptr, call_ptr, iter_ptr, M_VIA_SIZE,  *
*               M2_VIA_SIZE, C_VIA_SIZE, D_VIA_SIZE, P_VIA_SIZE,         *
*               V_VIA_SIZE                                               *
*       Modules Called:  Out_File, Out_Port_Tree, Out_Rect_Tree,         *
*               Out_Wire_Tree, Out_Via_Tree, Out_Call_Tree,              *
*               Out_Iter_Tree                                            *
*       Calling Modules:  Main (7.0)                                     *
*       Function:  Output the complete cell layout to disk               *
*       PDL:                                                             *
*         If there are ports defined                                     *
*           Output the ports;                                            *
*         If there are rectangles defined                                *
*           Output the rectangles;                                       *
*         If there are wires statements defined                          *
*           Output the wire statements in comments,                      *
*           Output the rectangles defining the wires,                    *
*           Output the vias defining the wires;                          *
*         If there are vias defined                                      *
*           Output the vias;                                             *
*         If there are call statements defined                           *
*           Output the call statements;                                  *
*         If there are iterate statements defined                        *
*           Output the iterations.                                       *
*************************************************************************/
Out_Cell_Layout (fp)
FILE    *fp;
{
printf ("\n(START PORT DEFINITIONS);");
Out_File ("\n(START PORT DEFINITIONS);", "\n", fp);
if (port_ptr != NULL)
  Out_Port_Tree (port_ptr, fp);
Out_File ("(END PORT DEFINITIONS);", "\n", fp);
printf ("\n(START RECTANGLE DEFINITIONS);");
Out_File ("\n(START RECTANGLE DEFINITIONS);", "\n", fp);
if (f_rect != NULL)
  {
  Out_File ("(rectangles in first level metal);", "\nL CM;\n", fp);
  Out_Rect_Tree (f_rect, fp);
  }
if (s_rect != NULL)
  {
  Out_File ("(rectangles in second level metal);", "\nL CM2;\n", fp);
  Out_Rect_Tree (s_rect, fp);
  }
```

```
if (d_rect != NULL)
  {
  Out_File ("(rectangles in diffusion);", "\nL CD;\n", fp);
  Out_Rect_Tree (d_rect, fp);
  }
if (p_rect != NULL)
  {
  Out_File ("(rectangles in polysilicon);", "\nL CP;\n", fp);
  Out_Rect_Tree (p_rect, fp);
  }
Out_File ("(END RECTANGLE DEFINITIONS);", "\n", fp);
printf ("\n(START WIRE DEFINITIONS);");
Out_File ("\n(START WIRE DEFINITIONS);", "\n", fp);
if (wire_ptr != NULL)
  Out_Wire_Tree (wire_ptr, fp);
if (w_f_rect != NULL)
  {
  Out_File ("L CM;", "\n", fp);
  Out_Rect_Tree (w_f_rect, fp);
  }
if (w_s_rect != NULL)
  {
  Out_File ("L CM2;", "\n", fp);
  Out_Rect_Tree (w_s_rect, fp);
  }
if (w_d_rect != NULL)
  {
  Out_File ("L CD;", "\n", fp);
  Out_Rect_Tree (w_d_rect, fp);
  }
if (w_p_rect != NULL)
  {
  Out_File ("L CP;", "\n", fp);
  Out_Rect_Tree (w_p_rect, fp);
  }
if (w_s_via != NULL)
  {
  Out_File ("L CM;", "\n",fp);
  Out_Via_Tree(w_s_via, M_VIA_SIZE, fp);
  Out_File ("L CM2;", "\n",fp);
  Out_Via_Tree(w_s_via, M2_VIA_SIZE, fp);
  Out_File ("L CC;", "\n",fp);
  Out_Via_Tree(w_s_via, C_VIA_SIZE, fp);
  }
if (w_d_via != NULL)
  {
  Out_File ("L CM;", "\n",fp);
  Out_Via_Tree(w_d_via, M_VIA_SIZE, fp);
  Out_File ("L CD;", "\n",fp);
  Out_Via_Tree(w_d_via, D_VIA_SIZE, fp);
  Out_File ("L CV;", "\n",fp);
  Out_Via_Tree(w_d_via, V_VIA_SIZE, fp);
  }
if (w_p_via != NULL)
  {
  Out_File ("L CM;", "\n",fp);
  Out_Via_Tree(w_p_via, M_VIA_SIZE, fp);
  Out_File ("L CP;", "\n",fp);
  Out_Via_Tree(w_p_via, P_VIA_SIZE, fp);
  Out_File ("L CV;", "\n",fp);
  Out_Via_Tree(w_p_via, V_VIA_SIZE, fp);
  }
Out_File ("(END WIRE DEFINITIONS);", "\n", fp);
printf ("\n(START VIA DEFINITIONS);");
Out_File ("\n(START VIA DEFINITIONS);", "\n", fp);
if (s_via != NULL)
  {
  Out_File ("(vias in Second to first metal first m layer);","\nL CM;\n",fp);
  Out_Via_Tree(s_via, M_VIA_SIZE, fp);
  Out_File ("(vias in second to first metal second m layer);","\nL CM2;\n",fp);
  Out_Via_Tree(s_via, M2_VIA_SIZE, fp);
  Out_File ("(vias in second to first metal cut layer);","\nL CC;\n",fp);
  Out_Via_Tree(s_via, C_VIA_SIZE, fp);
  }
if (d_via != NULL)
  {
  Out_File ("(vias in Diff to first metal first m layer);","\nL CM;\n",fp);
  Out_Via_Tree(d_via, M_VIA_SIZE, fp);
  Out_File ("(vias in diff to first metal diff layer);","\nL CD;\n",fp);
  Out_Via_Tree(d_via, D_VIA_SIZE, fp);
  Out_File ("(vias in diff to first metal cut layer);","\nL CV;\n",fp);
  Out_Via_Tree(d_via, V_VIA_SIZE, fp);
  }
if (p_via != NULL)
  {
  Out_File ("(vias in Poly to first metal first m layer);","\nL CM;\n",fp);
  Out_Via_Tree(p_via, M_VIA_SIZE, fp);
  Out_File ("(vias in poly to first metal poly layer);","\nL CP;\n",fp);
  Out_Via_Tree(p_via, P_VIA_SIZE, fp);
  Out_File ("(vias in poly to first metal cut layer);","\nL CV;\n",fp);
  Out_Via_Tree(p_via, V_VIA_SIZE, fp);
  }
Out_File ("(END VIA DEFINITIONS);", "\n", fp);
printf ("\n(START CALL STATEMENTS);");
Out_File ("\n(START CALL STATEMENTS);", "\n", fp);
if (call_ptr != NULL)
  Out_Call_Tree (call_ptr, fp);
Out_File ("(END CALL STATEMENTS);", "\n", fp);
printf ("\n(START ITER STATEMENTS);");
Out_File ("\n(START ITER STATEMENTS);", "\n", fp);
if (iter_ptr != NULL)  Out_Iter_Tree (iter_ptr, fp);
Out_File ("(END ITER STATEMENTS);", "\n", fp);
}
#endif
```

```
#ifneed Out_Port_Tree
/************************************************************************
*                                                                       *
*       Module Name:  Out_Port_Tree                                     *
*       Module Number:  7.3.1                                           *
*       Version/Date:  1.0   28 July 1985                               *
*       Input Parameters:  port, fp                                     *
*       Output Parameters:  none                                        *
*       Globals Used:  none                                             *
*       Modules Called:  Out_File, Out_Port_Tree (recursion)            *
*       Calling Modules:  Out_Cell_Layout, Out_Port_Tree                *
*       Function:  Place ports on disk                                  *
*       PDL:                                                            *
*         If the location is defined                                    *
*           If the port hasn't been deleted                             *
*             Output the CIF for the port;                              *
*           Go to the next port location.                               *
*                                                                       *
************************************************************************/

Out_Port_Tree (p_ptr, fp)

struct inter_ports *p_ptr;
FILE    *fp;

{
char    a_num[7];

if (p_ptr != NULL)
  {
  if (p_ptr->y_loc != -9999)
    {
    Out_File ("94 ", p_ptr->port_name, fp);
    putc(' ', fp);
    itoa (p_ptr->x_loc, a_num);
    Out_File (a_num, ",", fp);
    itoa (p_ptr->y_loc, a_num);
    Out_File (a_num, " ", fp);
    if (p_ptr->port_layer == 'F')
      Out_File ("CM;", "\n", fp);
    else if (p_ptr->port_layer == 'S')
      Out_File ("CM2;", "\n", fp);
    else if (p_ptr->port_layer == 'D')
      Out_File ("CD;", "\n", fp);
    else if (p_ptr->port_layer == 'P')
      Out_File ("CP;", "\n", fp);
    }
  Out_Port_Tree (p_ptr->next_port, fp);
  }
}

#endif
```

```
#ifneed Out_Rect_Tree
/*************************************************************************
*       Module Name:  Out_Rect_Tree                                      *
*       Module Number:  7.3.2                                            *
*       Version/Date:  1.0   28 July 1985                                *
*       Input Parameters:  rect, fp                                      *
*       Output Parameters:  none                                         *
*       Globals Used:  min_x_b, min_y_b, max_x_b, max_y_b                *
*       Modules Called:  Out_File, Out_Rect_Tree (recursion)             *
*       Calling Modules:  Out_Cell_Layout, Out_Rect_Tree                 *
*       Function:  Place rectangles on disk                              *
*       PDL:                                                             *
*         If the location is defined                                     *
*           Output the left rectangle structure,                         *
*           If the rectangle hasn't been deleted                         *
*             Output the CIF for the rectangle;                          *
*             If needed adjust the cell bounds;                          *
*           Output the middle rectangle structure,                       *
*           Output the right rectangle structure.                        *
*************************************************************************/
Out_Rect_Tree (rect, fp)
struct rectangle *rect;
FILE    *fp;
{
char    a_num [7];
if (rect != NULL)
  {
  Out_Rect_Tree (rect->less_than_x, fp);
  if (rect->height != 0 || rect->length != 0)
    {
    Out_File ("B", " ", fp);
    itoa (rect->length, a_num);
    Out_File (a_num, " ", fp);
    itoa (rect->height, a_num);
    Out_File (a_num, " ", fp);
    itoa ((rect->x_location + rect->length/2), a_num);
    Out_File (a_num, ",", fp);
    itoa ((rect->y_location + rect->height/2), a_num);
    Out_File (a_num, ";\n", fp);
    if (rect->x_location < min_x_b)
      min_x_b = rect->x_location;
    if (rect->x_location + rect->length > max_x_b)
        max_x_b = rect->x_location + rect->length;
    if (rect->y_location < min_y_b)
      min_y_b = rect->y_location;
    if (rect->y_location + rect->height > max_y_b)
        max_y_b = rect->y_location + rect->height;
    }
  Out_Rect_Tree (rect->equals_x, fp);
  Out_Rect_Tree (rect->greater_than_x, fp);
  }}
#endif
```

```
#ifneed Out_Wire_Tree
/***********************************************************************
*                                                                      *
*       Module Name:  Out_Wire_Tree                                    *
*       Module Number:  7.3.3                                          *
*       Version/Date:  1.0  5 August 1985                              *
*       Input Parameters:  ptr, fp                                     *
*       Output Parameters:  none                                       *
*       Globals Used:  none                                            *
*       Modules Called:  Out_W_Segment, Out_Wire_Tree (recursion)      *
*                       Out_File                                       *
*       Calling Modules:  Out_Cell_Layout, Out_Wire_Tree               *
*       Function:  Output the wire statements to disk                  *
*       PDL:                                                           *
*         If there is a wire on this layer                             *
*           Walk the trinary tree to the left;                         *
*           If the wire has segments defined                           *
*             Output the starting location for the wire;               *
*             Output the wire segments recursively;                    *
*           Walk the tree to the middle;                               *
*           Walk the tree to the right;                                *
*                                                                      *
***********************************************************************/

Out_Wire_Tree (ptr, fp)

struct wire *ptr;
FILE    *fp;

{
char    a_num [7];

if (ptr != NULL)
  {
  Out_Wire_Tree (ptr->w_l_t_x, fp);
  if (ptr->w_param != NULL)
    {
    itoa (ptr->w_x_location, a_num);
    Out_File ("(Wire ", a_num, fp);
    itoa (ptr->w_y_location, a_num);
    Out_File (",", a_num, fp);
    putc (' ', fp);
    Out_W_Comment (0, ' ', ptr->w_param, fp);
    Out_File (");", "\n", fp);
    Out_W_Segment (ptr->w_x_location, ptr->w_y_location,ptr->w_param);
    }
  Out_Wire_Tree (ptr->w_e_x, fp);
  Out_Wire_Tree (ptr->w_g_t_x, fp);
  }
}
#endif
```

```
#ifneed Out_W_Comment
/************************************************************************
*                                                                       *
*       Module Name:  Out_W_Comment                                     *
*       Module Number:  7.3.3.1                                         *
*       Version/Date:  1.0   5 August 1985                              *
*       Input Parameters:  p_width, p_layer, seg, fp                    *
*       Output Parameters:  none                                        *
*       Globals Used:  none                                             *
*       Modules Called:  Out_File, Out_W_Comment (recursion)            *
*       Calling Modules:  Out_Wire_Tree, Out_W_Comment                  *
*       Function:  Output the segments of the wire to disk              *
*       PDL:                                                            *
*         If the location is defined                                    *
*           Output the comment describing this segment;                 *
*             Output the next wire segment.                             *
*                                                                       *
************************************************************************/

Out_W_Comment (p_width, p_layer, seg, fp)

int     p_width;
char    p_layer;
struct wire_parameters *seg;
FILE    *fp;

{
char    a_w [7],
        a_l [7];

if (seg != NULL)
  {
  if (p_layer != seg->w_layer)
    {
    putc (seg->w_layer,fp);
    putc (' ', fp);
    }
  if (p_width != seg->width)
    {
    itoa (seg->width, a_w);
    Out_File ("W", a_w, fp);
    putc (' ', fp);
    }
  putc (seg->direction, fp);
  itoa (seg->length, a_l);
  Out_File ("", a_l, fp);
  putc (' ', fp);
  Out_W_Comment (seg->width, seg->w_layer, seg->next_segment, fp);
  }
}
#endif
```

```
#ifneed Out_W_Segment
/************************************************************************
*                                                                       *
*       Module Name:  Out_W_Segment                                     *
*       Module Number:  7.3.3.2                                         *
*       Version/Date:  1.0   5 August 1985                              *
*       Input Parameters:  p_x, p_y, seg                                *
*       Output Parameters:  none                                        *
*       Globals Used:  w_f_rect, w_s_rect, w_p_rect, s_d_rect, w_s_via  *
*               w_p_via, w_d_via, M_VIA_SIZE                            *
*       Modules Called:  Out_File, Out_W_Segment (recursion)            *
*       Calling Modules:  Out_Wire_Tree, Out_W_Segment                  *
*       Function:  Output the segments of the wire to disk              *
*       PDL:                                                            *
*         If the location is defined                                    *
*           Set x,y to the bottom left of this rectangle;               *
*           Place the rectangle in memory;                              *
*           If there is a next segment                                  *
*             Set x,y to other end of present segment,                  *
*             If the next segment is on a different layer               *
*               Place a via in memory at this location;                 *
*             Set x,y location to fill in corner;                       *
*             Go to the next wire segment.                              *
*                                                                       *
************************************************************************/

Out_W_Segment (p_x, p_y, seg)

struct wire_parameters *seg;

{
int     v_x, v_y, x_loc, y_loc, r_height, r_length;
char    v_layer;

if (seg != NULL)
  {
  switch (seg->direction)
    {
    case 'f' :  x_loc = p_x;  y_loc = p_y - seg->width/2;
                r_height = seg->width;  r_length = seg->length;  break;
    case 'b' :  x_loc = p_x - seg->length;  y_loc = p_y - seg->width/2;
                r_height = seg->width;  r_length = seg->length;  break;
    case 'u' :  x_loc = p_x - seg->width/2;  y_loc = p_y;
                r_height = seg->length;  r_length = seg->width;  break;
    case 'd' :  x_loc = p_x - seg->width/2;  y_loc = p_y - seg->length;
                r_height = seg->length;  r_length = seg->width;  break;
    }
```

```
switch (seg->w_layer)
  {
  case 'F' :  Rect_Tree (r_height,r_length,x_loc,y_loc,&w_f_rect);
              break;
  case 'S' :  Rect_Tree (r_height,r_length,x_loc,y_loc,&w_s_rect);
              break;
  case 'D' :  Rect_Tree (r_height,r_length,x_loc,y_loc,&w_d_rect);
              break;
  case 'P' :  Rect_Tree (r_height,r_length,x_loc,y_loc,&w_p_rect);
              break;
  }


if (seg->next_segment != NULL)
  {
  switch (seg->direction)
    {
    case 'f' :  p_x = p_x + seg->length; break;
    case 'b' :  p_x = p_x - seg->length; break;
    case 'u' :  p_y = p_y + seg->length; break;
    case 'd' :  p_y = p_y - seg->length; break;
    }

  if (seg->w_layer != seg->next_segment->w_layer)
    {
    v_x = p_x - M_VIA_SIZE/2;
    v_y = p_y - M_VIA_SIZE/2;
    if (seg->w_layer == 'F')
      v_layer = seg->next_segment->w_layer;
      else v_layer = seg->w_layer;
    switch (v_layer)
      {
      case 'S' :  Via_Tree (v_x, v_y, &w_s_via);  break;
      case 'D' :  Via_Tree (v_x, v_y, &w_d_via);  break;
      case 'P' :  Via_Tree (v_x, v_y, &w_p_via);  break;
      }
    }
    switch (seg->next_segment->direction)
      {
      case 'f' :
        if (seg->direction == 'u' || seg->direction == 'd')
          {
          p_x = p_x - seg->width/2;
          seg->next_segment->length = seg->next_segment->length + seg->width/2;
          }
        break;
      case 'b' :
        if (seg->direction == 'u' || seg->direction == 'd')
          {
          p_x = p_x + seg->width/2;
          seg->next_segment->length = seg->next_segment->length + seg->width/2;
          }
        break;
      case 'u' :
        if (seg->direction == 'f' || seg->direction == 'b')
          {
          p_y = p_y - seg->width/2;
          seg->next_segment->length = seg->next_segment->length + seg->width/2;
          }
        break;
      case 'd' :
        if (seg->direction == 'f' || seg->direction == 'b')
          {
          p_y = p_y + seg->width/2;
          seg->next_segment->length = seg->next_segment->length + seg->width/2;
          }
        break;
      }
    }
  Out_W_Segment (p_x, p_y, seg->next_segment);
  }
}
#endif
```

```
#ifneed Out_Via_Tree
/************************************************************************
*       Module Name:  Out_Via_Tree                                      *
*       Module Number:  7.3.4                                           *
*       Version/Date:  1.0   5 August 1985                              *
*       Input Parameters:  l_via, via_size, fp                          *
*       Output Parameters:  none                                        *
*       Globals Used:  min_x_b, min_y_b, max_x_b, max_y_b               *
*       Modules Called:  Out_File, Out_Via_Tree (recursion)             *
*       Calling Modules:  Out_Cell_Layout, Out_Via_Tree                 *
*       Function:  Output the vias to disk one rectangle at a time      *
*       PDL:                                                            *
*         If this location is defined                                   *
*           Output the left via tree;                                   *
*           If this via hasn't been deleted                             *
*             Output the box for the via,                               *
*             If needed adjust the cell bounds;                         *
*           Output the middle via tree;                                 *
*           Output the right via tree.                                  *
************************************************************************/
Out_Via_Tree (l_via, via_size, fp)
int     via_size;
struct via *l_via;
FILE    *fp;
{
char    a_num [7];
if (l_via != NULL)
  {
  Out_Via_Tree (l_via->v_l_t_x, via_size, fp);
  if (l_via->v_y_location != -9999)
    {
    Out_File ("B", " ", fp);
    itoa (via_size, a_num);
    Out_File (a_num, " ", fp);
    Out_File (a_num, " ", fp);
    itoa (l_via->v_x_location + M_VIA_SIZE/2, a_num);
    Out_File (a_num, ",", fp);
    itoa (l_via->v_y_location + M_VIA_SIZE/2, a_num);
    Out_File (a_num, ";\n", fp);
    if (l_via->v_x_location < min_x_b)
      min_x_b = l_via->v_x_location;
    if (l_via->v_x_location + via_size > max_x_b)
      max_x_b = l_via->v_x_location + via_size;
    if (l_via->v_y_location < min_y_b)
      min_y_b = l_via->v_y_location;
    if (l_via->v_y_location + via_size > max_y_b)
      max_y_b = l_via->v_y_location + via_size;
    }
  Out_Via_Tree (l_via->v_e_x, via_size, fp);
  Out_Via_Tree (l_via->v_g_t_x, via_size, fp);
  }}
#endif
```

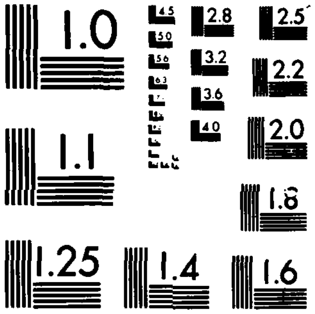
```
#ifneed Out_Call_Tree
/************************************************************************
*                                                                       *
*       Module Name:  Out_Call_Tree                                     *
*       Module Number:  7.3.5                                           *
*       Version/Date:  1.0   21 August 1985                             *
*       Input Parameters:  ptr, fp                                      *
*       Output Parameters:  none                                        *
*       Globals Used:  none                                             *
*       Modules Called:  Out_File, Out_Call_Tree (recursion),           *
*               Out_C_Statements, Out_P_Names                           *
*       Calling Modules:  Out_Cell_Layout, Out_Call_Tree                *
*       Function:  Output the Call statements to disk                   *
*       PDL:                                                            *
*         If this location is defined                                   *
*           Output the left call tree;                                  *
*           If this call hasn't been deleted                            *
*             Output the comment describing this call,                  *
*             Output the CIF call statement,                            *
*             Output the portnames;                                     *
*           Output the middle call tree;                                *
*           Output the right call tree.                                 *
*                                                                       *
************************************************************************/
Out_Call_Tree (ptr, fp)
struct call_statement  *ptr;
FILE    *fp;
{
char    a_num [7];

if (ptr != NULL)
  {
  Out_Call_Tree (ptr->l_t_name, fp);
  if (ptr->tran_y != -9999)
    {
    Out_File ("(Call ", ptr->cellname, fp);
    itoa (ptr->tran_x, a_num);  Out_File (" ", a_num, fp);
    itoa (ptr->tran_y, a_num);  Out_File (",", a_num, fp);
    itoa (ptr->x_min, a_num);  Out_File (" bounds ", a_num, fp);
    itoa (ptr->y_min, a_num);  Out_File (",", a_num, fp);
    itoa (ptr->x_max, a_num);  Out_File (" ", a_num, fp);
    itoa (ptr->y_max, a_num);  Out_File (",", a_num, fp);
    Out_File (");", "\n", fp);
    Out_C_Statement (ptr->cifnum, ptr->tran_x,ptr->tran_y,ptr->mir_x,
        ptr->mir_y,ptr->rot_x,ptr->x_min,ptr->y_min,ptr->x_max,ptr->y_max,fp);
    Out_P_Names (ptr->c_ports,fp);
    }
  Out_Call_Tree (ptr->e_name, fp);
  Out_Call_Tree (ptr->g_t_name, fp);
  }
}
#endif
```

AD-A164 109   A MICROCOMPUTER-BASED MENU-DRIVEN CHIP LAYOUT LANGUAGE   4/4
(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH
SCHOOL OF ENGINEERING S C MORRESE DEC 85
UNCLASSIFIED AFIT/GCS/ENG/85D-11   F/G 9/5   NL

END

FILMED

DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```
#ifneed Out_C_Statement
/*************************************************************************
*                                                                        *
*       Module Name:  Out_C_Statement                                    *
*       Module Number:  7.3.5.1                                          *
*       Version/Date:  1.0   21 August 1985                              *
*       Input Parameters:  ptr, fp                                       *
*       Output Parameters:  none                                         *
*       Globals Used:  min_x_b, min_y_b, max_x_b, max_y_b                *
*       Modules Called:  Out_File                                        *
*       Calling Modules:  Out_Call_Tree, Out_Iter_Tree                   *
*       Function:  Output the Call statements to disk                    *
*       PDL:                                                             *
*             Output the CIF call statement,                             *
*             If needed adjust the bounds,                               *
*                                                                        *
*************************************************************************/
Out_C_Statement (cifnum, tran_x, tran_y, mir_x, mir_y, rot_x, x_min, y_min,
                 x_max, y_max, fp)
FILE    *fp;
{
char    a_num [7];
int     temp;

itoa (cifnum, a_num);    Out_File ("C ", a_num, fp);
if (mir_x == TRUE)
  Out_File (" M", " X", fp);
if (mir_y == TRUE)
  Out_File (" M", " Y", fp);
if (rot_x != ' ')
  {
  if (rot_x == '3')  Out_File (" R ", "1,0", fp);
  else if (rot_x == '6') Out_File (" R ", "0,-1", fp);
  else if (rot_x == '9') Out_File (" R ", "-1,0", fp);
  else if (rot_x == '1') Out_File (" R ", "0,1", fp);
  }
if (mir_x == TRUE)
  {
  temp = y_min;
  y_min = -y_max;
  y_max = -temp;
  }
if (mir_y == TRUE)
  {
  temp = x_min;
  x_min = -x_max;
  x_max = -temp;
  }
if (rot_x == '6')
  {
  temp = x_min;
  x_min = y_min;
  y_min = -x_max;
  x_max = y_max;
  y_max = - temp;
  }
else if (rot_x == '9')
  {
  temp = x_min;
  x_min = -x_max;
  x_max = -x_min;
  temp = y_min;
  y_min = -y_max;
  y_max = -y_min;
  }
else if (rot_x == '1')
  {
  temp = x_min;
  x_min = -y_max;
  y_max = x_max;
  x_max = -y_min;
  y_min = temp;
  }
itoa ((tran_x - x_min), a_num);    Out_File (" T ", a_num, fp);
itoa ((tran_y - y_min), a_num);    Out_File (",", a_num, fp);
Out_File (";", "\n", fp);
if (tran_x < min_x_b) min_x_b = tran_x;
if ((tran_x + x_max - x_min) > max_x_b) max_x_b = tran_x + x_max - x_min;
if (tran_y < min_y_b) min_y_b = tran_y;
if (tran_y + y_max - y_min > max_y_b) max_y_b = tran_y + y_max -y_min;
}

#endif
```

```
#ifneed Out_P_Names
/************************************************************************
*                                                                       *
*       Module Name:  Out_P_Names                                       *
*       Module Number:  7.3.5.2                                         *
*       Version/Date:  1.0   21 August 1985                             *
*       Input Parameters:  ptr, fp                                      *
*       Output Parameters:  none                                        *
*       Globals Used:  none                                             *
*       Modules Called:  Out_File                                       *
*       Calling Modules:  Out_Call_Tree, Out_Iter_Tree                  *
*       Function:  Output the Call statements to disk                   *
*       PDL:                                                            *
*             While there are portnames associated with this call      *
*               Output the portnames;                                   *
*                                                                       *
************************************************************************/

Out_P_Names (ptr, fp)

struct named_ports *ptr;
FILE    *fp;

{
while (ptr != NULL)
  {
  if (ptr->port_name[0] != '-')
    {
    Out_File ("(Port ", ptr->port_name, fp);
    Out_File (");", "\n", fp);
    }
  ptr = ptr->next_name;
  }
}

#endif
```

```
#ifneed Out_Iter_Tree
/*************************************************************************
*                                                                        *
*       Module Name:  Out_Iter_Tree                                      *
*       Module Number:  7.3.6                                            *
*       Version/Date:  1.0   17 September 1985                           *
*       Input Parameters:  ptr, fp                                       *
*       Output Parameters:  none                                         *
*       Globals Used:  none                                              *
*       Modules Called:  Out_File, Out_Iter_Tree (recursion),            *
*               Out_C_Statement, Out_P_Names                             *
*       Calling Modules:  Out_Cell_Layout, Out_Iter_Tree                 *
*       Function:  Output the Iter statements to disk                    *
*       PDL:                                                             *
*         If this location is defined                                    *
*           Output the left call tree;                                   *
*           If this call hasn't been deleted                             *
*             Output the comment describing this call,                   *
*             Output the CIF call statement,                             *
*             If needed adjust the bounds,                               *
*             While there are portnames associated with this call        *
*               Output the portnames;                                    *
*           Output the middle call tree;                                 *
*           Output the right call tree.                                  *
*                                                                        *
*************************************************************************/

Out_Iter_Tree (ptr, fp)

struct iter_statement  *ptr;
FILE    *fp;

{
char    a_num [7];
int     i, j, t_x, t_y;

if (ptr != NULL)
  {
  Out_Iter_Tree (ptr->l_t_cifnum, fp);
  if (ptr->x_iters != -9999)
    {
    Out_File ("(Iter ", ptr->cellname, fp);
    putc (' ', fp);
    itoa (ptr->x_iters, a_num);  Out_File (a_num, ",", fp);
    itoa (ptr->y_iters, a_num);  Out_File (a_num, " ", fp);
    itoa (ptr->x_pitch, a_num);  Out_File (a_num, ",", fp);
    itoa (ptr->y_pitch, a_num);  Out_File (a_num, " ", fp);
    itoa (ptr->i_call->tran_x, a_num);  Out_File (a_num, ",", fp);
    itoa (ptr->i_call->tran_y, a_num);  Out_File (a_num, " ", fp);
    itoa (ptr->i_call->x_min, a_num);  Out_File (" bounds ", a_num, fp);
    itoa (ptr->i_call->y_min, a_num);  Out_File (",", a_num, fp);
    itoa (ptr->i_call->x_max, a_num);  Out_File (" ", a_num, fp);
    itoa (ptr->i_call->y_max, a_num);  Out_File (",", a_num, fp);
    Out_File (");", "\n", fp);
    i = 0;
    while (i < ptr->x_iters)
      {
      j = 0;
      while (j < ptr->y_iters)
        {
        t_x = ptr->i_call->tran_x + i * ptr->x_pitch;
        t_y = ptr->i_call->tran_y + j * ptr->y_pitch;
        Out_C_Statement (ptr->cifnum, t_x,t_y,ptr->i_call->mir_x,
          ptr->i_call->mir_y,ptr->i_call->rot_x,ptr->i_call->x_min,
          ptr->i_call->y_min,ptr->i_call->x_max,ptr->i_call->y_max,fp);
        ++j;
        }
      ++i;
      }
    Out_P_Names (ptr->i_call->c_ports,fp);
    }
  Out_Iter_Tree (ptr->e_cifnum, fp);
  Out_Iter_Tree (ptr->g_t_cifnum, fp);
  }
}

#endif
```

```
#ifneed Footer_Out_File
/***********************************************************************
*                                                                      *
*       Module Name:  Footer_Out_File                                  *
*       Module Number:  7.4                                            *
*       Version/Date:  1.0   28 July 1985                              *
*       Input Parameters:  cellname, a_cifnum, fp                      *
*       Output Parameters:  none                                       *
*       Globals Used:  min_x_b, min_y_b, max_x_b, max_y_b              *
*       Modules Called:  Out_File                                      *
*       Calling Modules:  SAVECELL Main                                *
*       Funtion:  Output the CIF file footer                           *
*       PDL:  N/A                                                      *
*                                                                      *
***********************************************************************/

Footer_Out_File (cellname, a_cifnum, fp)
char    *cellname,
        *a_cifnum;
FILE    *fp;
{
char    a_num [7];

printf ("attaching file footer for cell %s\n", cellname);
Out_File ("(bounds", " ", fp);
Out_File (cellname, " ", fp);
itoa (min_x_b, a_num);
Out_File (a_num, ",", fp);
itoa (min_y_b, a_num);
Out_File (a_num, " ", fp);
itoa (max_x_b, a_num);
Out_File (a_num, ",", fp);
itoa (max_y_b, a_num);
Out_File (a_num, ");\nDF;\nC ",fp);
Out_File (a_cifnum, " ", fp);
itoa (-min_x_b, a_num);
Out_File ("T ", a_num, fp);
itoa (-min_y_b, a_num);
Out_File (",", a_num, fp);
Out_File (";\n", "E\n",fp);
}

#endif
```

Appendix C

Sample MDCLL Output

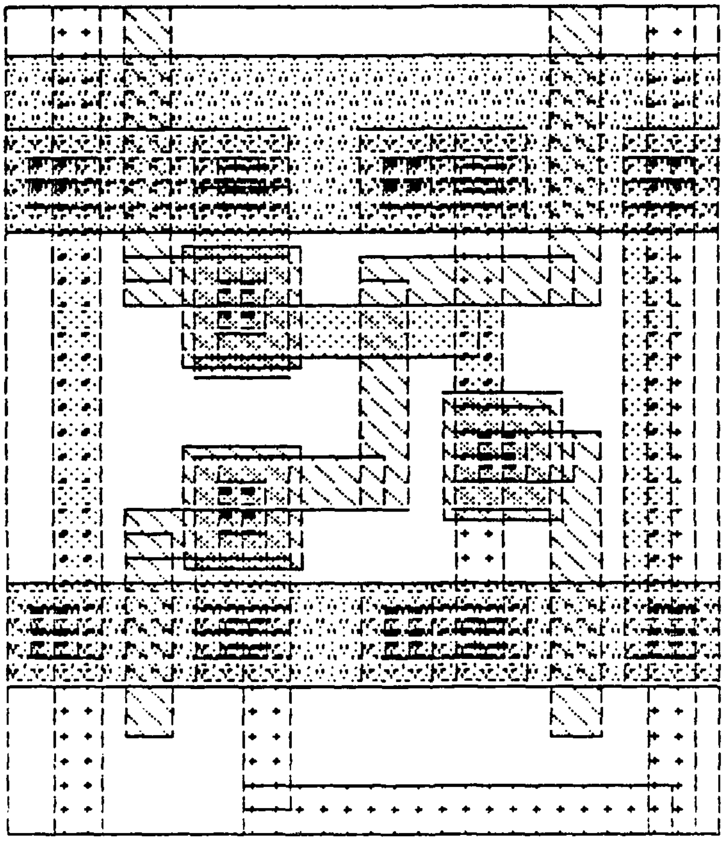
Included in this appendix are the CIF files for six cells created using the MDCLL system. Each cell is followed by a plot of the circuit described using the program MCIFPLOT. The cells included are MASK (used to mask bits in a content addressable memory, CAM), MEM (a six transistor memory cell), SUB (a full subtractor for use in a CAM), BINBOUT (a combination of MASK, MEM, SUB, and needed interconnections), MEM2 (two calls to the BINBOUT cell), and MEM8 (an iteration of eight BINBOUT cells). The finalized CIF file version (as provided by the FINALIZE program) of the BINBOUT, MEM2, and MEM8 cells is not provided since each is simply a combination of the called cells CIF descriptions. However, the finalized version was used to produce the plots for each of these cells.

```
DS 32077 50 1;
(TITLE MASK 32077);

(START PORT DEFINITIONS);
(END PORT DEFINITIONS);

(START RECTANGLE DEFINITIONS);
(END RECTANGLE DEFINITIONS);

(START WIRE DEFINITIONS);
(Wire 0,55 D W14 f60 );
(Wire 0,16 D W8 f60 );
(Wire 4,16 D W4 u0 F f2 u36 b2 D u0 );
(Wire 6,0 S W4 u66 );
(Wire 12,66 P W4 d22 f8 d2 F d2 f20 d10 f2 P f6 d22 );
(Wire 12,8 P W4 u16 f8 u2 F u0 P u2 f12 u16 f16 u22 );
(Wire 20,37 F W8 u15 S b12 );
(Wire 20,26 F W8 d10 S W4 f2 d14 f34 u64 );
(Wire 34,16 D W4 b0 F f6 S u36 F b6 D b0 );
(Wire 56,16 D W4 b0 F b2 u36 f2 D u0 );
L CM;
B 4 4 4,16;
B 4 38 6,33;
B 4 4 6,52;
B 8 15 20,44;
B 8 10 20,21;
B 4 2 20,41;
B 22 4 29,40;
B 4 0 20,26;
B 6 4 37,16;
B 8 4 38,52;
B 4 12 40,36;
B 4 4 40,30;
B 4 38 54,33;
B 4 4 54,52;
B 2 4 55,16;
L CS;
B 4 66 6,33;
B 16 8 16,52;
B 6 4 19,16;
B 4 16 22,10;
B 36 4 38,2;
B 4 38 40,33;
B 4 66 56,33;
```

```
L CD;
B 60 14 30,55;
B 60 8 30,16;
B 4 0 4,16;
B 4 2 4,51;
B 0 4 34,16;
B 0 4 34,52;
B 4 2 56,51;
B 0 4 56,16;
L CP;
B 4 22 12,55;
B 10 4 15,44;
B 4 16 12,16;
B 10 4 15,24;
B 4 4 20,44;
B 4 4 20,24;
B 4 2 20,27;
B 14 4 25,28;
B 4 18 32,35;
B 18 4 39,44;
B 6 4 45,30;
B 4 24 48,20;
B 4 24 48,54;
L CM;
B 8 8 20,52;
B 8 8 20,16;
B 8 8 40,16;
B 8 8 40,52;
L CS;
B 8 8 20,52;
B 8 8 20,16;
B 8 8 40,16;
B 8 8 40,52;
L CC;
B 4 4 20,52;
B 4 4 20,16;
B 4 4 40,16;
B 4 4 40,52;
L CM;
B 8 8 4,16;
B 8 8 4,52;
B 8 8 34,16;
B 8 8 34,52;
B 8 8 56,16;
B 8 8 56,52;
```

```
L CD;
B 8 8 4,16;
B 8 8 4,52;
B 8 8 34,16;
B 8 8 34,52;
B 8 8 56,16;
B 8 8 56,52;
L CG;
B 4 4 4,16;
B 4 4 4,52;
B 4 4 34,16;
B 4 4 34,52;
B 4 4 56,16;
B 4 4 56,52;
L CM;
B 8 8 20,42;
B 8 8 20,26;
B 8 8 20,26;
B 8 8 42,30;
L CP;
B 10 10 20,42;
B 10 10 20,26;
B 10 10 20,26;
B 10 10 42,30;
L CG;
B 4 4 20,42;
B 4 4 20,26;
B 4 4 20,26;
B 4 4 42,30;
(END WIRE DEFINITIONS);

(START VIA DEFINITIONS);
(END VIA DEFINITIONS);

(START CALL STATEMENTS);
(END CALL STATEMENTS);

(START ITER STATEMENTS);
(END ITER STATEMENTS);
(bounds MASK 0,0 60,66);
DF;
C 32077 T 0,0;
E
```

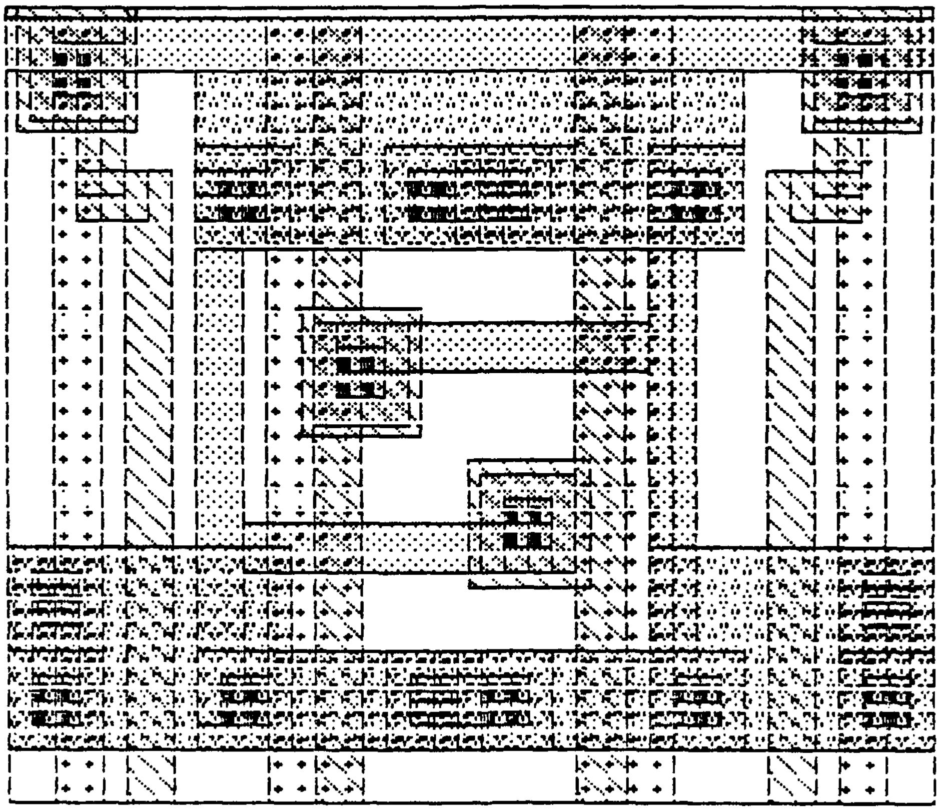
mcifplot* of MASK.CIF
Window: -3300 0 0 7150
Scale: 1 micron is .1118881 inches



```
DS 4139 50 1;
(TITLE MEM 4139);

(START PORT DEFINITIONS);
(END PORT DEFINITIONS);
(START RECTANGLE DEFINITIONS);
(END RECTANGLE DEFINITIONS);

(START WIRE DEFINITIONS);
(Wire 0,12 D W16 f24 );
(Wire 0,60 F W4 f78 );
(Wire 4,8 D W4 u0 F u8 S u0 );
(Wire 6,0 S W4 u62 );
(Wire 6,58 F W4 f0 P f2 d10 f4 d48 );
(Wire 16,51 D W14 f46 );
(Wire 20,8 D W4 u0 F b2 u40 f2 D u0 );
(Wire 20,20 F W4 f24 u2 P f0 );
(Wire 24,8 D W8 f30 );
(Wire 26,0 S W8 u62 );
(Wire 28,0 P W4 u62 );
(Wire 36,48 D W4 u0 F f6 S W8 f6 );
(Wire 42,8 D W4 u0 F b6 S W8 b6 );
(Wire 50,0 P W4 u62 );
(Wire 52,0 S W8 u62 );
(Wire 54,36 F W4 b24 d2 P b0 );
(Wire 58,8 D W4 u0 F b2 u40 f2 D u0 );
(Wire 72,0 S W4 u62 );
(Wire 72,58 F W4 f0 P b2 d10 b4 d48 );
(Wire 74,8 D W4 u0 F u8 S u0 );
(Wire 78,12 D W16 b24 );
L CM;
B 78 4 39,60;
B 4 8 4,12;
B 0 4 6,58;
B 4 42 18,27;
B 4 4 18,48;
B 4 4 20,8;
B 24 4 32,20;
B 4 4 30,36;
B 24 4 42,36;
B 8 4 38,48;
B 8 4 40,8;
B 4 4 44,20;
B 4 42 56,27;
B 4 4 56,48;
B 4 4 58,8;
B 0 4 72,58;
B 4 8 74,12;
L CS;
B 4 0 4,16;
B 4 62 6,31;
```

```
B 8 62 26,31;
B 6 8 33,8;
B 6 8 45,48;
B 8 62 52,31;
B 4 62 72,31;
B 4 0 74,16;
L CD;
B 24 16 12,12;
B 4 0 4,8;
B 46 14 39,51;
B 4 0 20,8;
B 4 2 20,47;
B 30 8 39,8;
B 4 0 36,48;
B 4 0 42,8;
B 24 16 66,12;
B 4 0 58,8;
B 4 2 58,47;
B 4 0 74,8;
L CP;
B 2 4 7,58;
B 4 12 8,54;
B 6 4 9,48;
B 4 50 12,25;
B 4 62 28,31;
B 2 4 31,34;
B 2 4 43,22;
B 4 62 50,31;
B 4 50 66,25;
B 6 4 69,48;
B 4 12 70,54;
B 2 4 71,58;
L CM;
B 8 8 4,16;
B 8 8 36,8;
B 8 8 42,48;
B 8 8 74,16;
L CS;
B 8 8 4,16;
B 8 8 36,8;
B 8 8 42,48;
B 8 8 74,16;
L CC;
B 4 4 4,16;
B 4 4 36,8;
B 4 4 42,48;
B 4 4 74,16;
L CM;
B 8 8 4,8;
B 8 8 20,8;
B 8 8 20,48;
B 8 8 36,48;
```

```
mcifplot* of MEM.CIF
Window: -3150 0 0 7150
Scale: 1 micron is .1118881 inches
```



```
B 8 8 42,8;
B 8 8 58,8;
B 8 8 58,48;
B 8 8 74,8;
L CD;
B 8 8 4,8;
B 8 8 20,8;
B 8 8 20,48;
B 8 8 36,48;
B 8 8 42,8;
B 8 8 58,8;
B 8 8 58,48;
B 8 8 74,8;
L CG;
B 4 4 4,8;
B 4 4 20,8;
B 4 4 20,48;
B 4 4 36,48;
B 4 4 42,8;
B 4 4 58,8;
B 4 4 58,48;
B 4 4 74,8;
L CM;
B 8 8 6,58;
B 8 8 30,34;
B 8 8 44,22;
B 8 8 72,58;
L CP;
B 10 10 6,58;
B 10 10 30,34;
B 10 10 44,22;
B 10 10 72,58;
L CG;
B 4 4 6,58;
B 4 4 30,34;
B 4 4 44,22;
B 4 4 72,58;
(END WIRE DEFINITIONS);

(START VIA DEFINITIONS);
(END VIA DEFINITIONS);

(START CALL STATEMENTS);
(END CALL STATEMENTS);

(START ITER STATEMENTS);
(END ITER STATEMENTS);
(bounds MEM 0,0 78,64);
DF;
C 4139 T 0,0;
E
```

```
DS 4888 50 1;
(TITLE SUB 4888);

(START PORT DEFINITIONS);
(END PORT DEFINITIONS);

(START RECTANGLE DEFINITIONS);
(END RECTANGLE DEFINITIONS);

(START WIRE DEFINITIONS);
(Wire 0,10 D W8 f44 u6 f8 u0 f42 );
(Wire 0,53 D W14 f94 );
(Wire 4,50 D W4 u0 F d2 f12 d10 f6 d2 P u6 f14 u22 );
(Wire 4,10 D W4 u0 F u2 f12 u8 f6 u2 P d4 f14 d16 );
(Wire 10,24 P W8 b6 F u10 S W4 u2 f40 );
(Wire 12,2 P W4 u62 );
(Wire 28,8 D W4 u0 F u2 f6 u38 b6 u2 D u0 );
(Wire 37,29 F W4 f0 P d2 f8 d2 f14 );
(Wire 45,2 S W4 u62 );
(Wire 47,15 S W4 b0 F f6 D b0 );
(Wire 47,50 S W4 b0 F f6 D f0 );
(Wire 61,8 P W4 u18 f6 u8 f10 u30 );
(Wire 69,10 D W4 u0 F u6 S u26 W0 f0 F b0 W4 u8 D u0 );
(Wire 77,8 P W4 u18 f12 );
(Wire 79,42 P W4 f32 );
(Wire 85,16 D W4 b0 F f6 S u34 F b6 D u0 );
(Wire 101,18 D W8 f16 );
(Wire 101,53 D W14 f24 );
(Wire 101,29 F W4 b12 u2 P u0 F d2 b32 u6 P f2 u6 f2 u22 );
(Wire 105,18 D W4 f0 F b2 u32 f2 D u0 );
(Wire 113,10 P W4 u54 );
(Wire 113,2 S W8 u62 );
(Wire 115,41 P W10 d10 );
(Wire 121,50 D W4 u0 F W8 d14 b6 S u0 );
(Wire 121,18 D W8 u0 F f8 S u0 );
(Wire 131,2 S W8 u62 );
L CM;
B 8 14 4,27;
B 4 2 4,49;
B 14 4 9,48;
B 4 2 4,11;
B 14 4 9,12;
B 4 12 16,44;
B 8 4 18,38;
B 4 10 16,15;
B 8 4 18,20;
B 4 4 22,38;
B 4 4 22,20;
B 4 2 28,9;
B 8 4 30,10;
B 4 4 28,48;
```

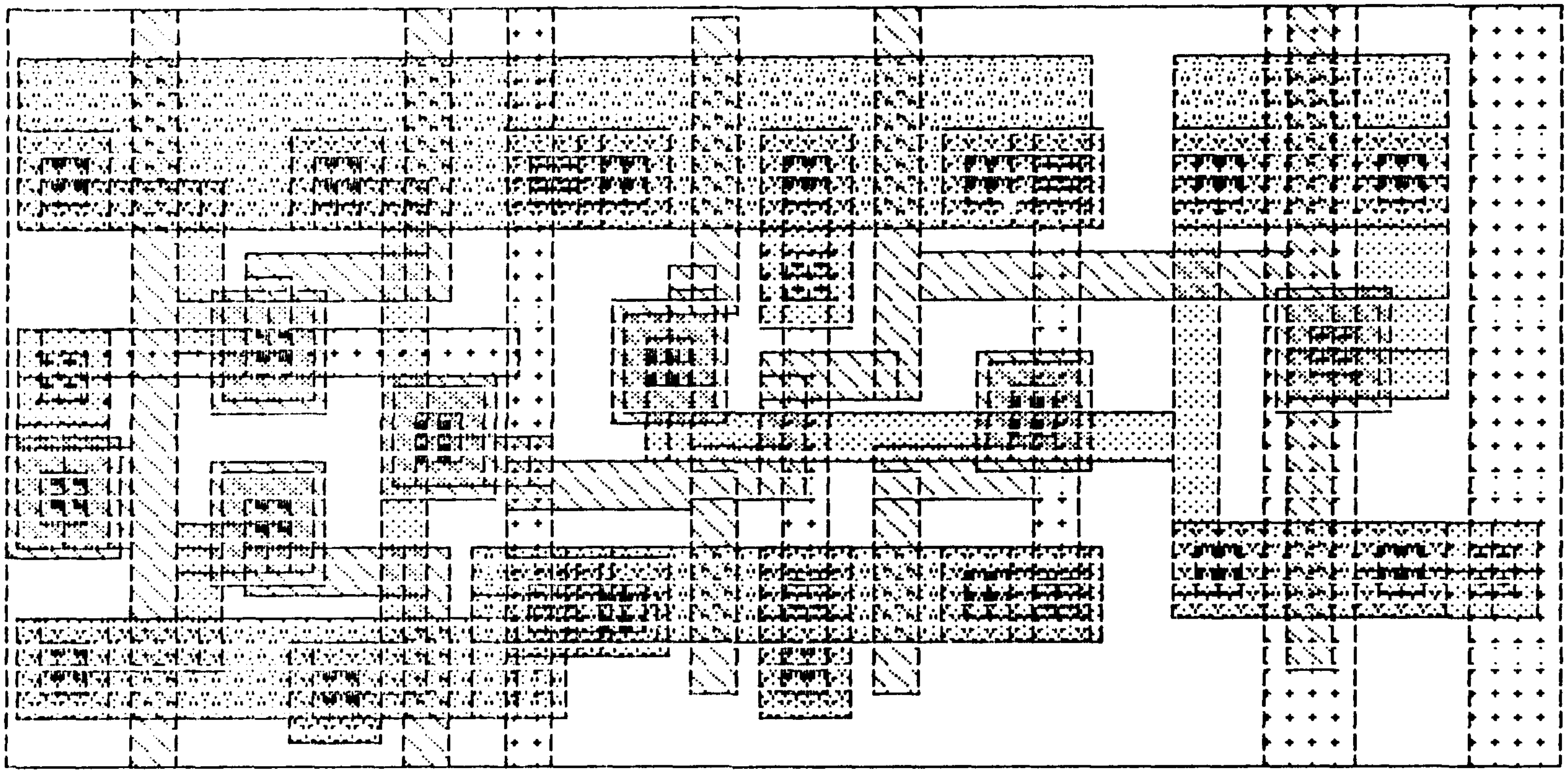
```
B 8 4 32,48;
B 4 40 34,28;
B 0 4 37,29;
B 6 4 50,15;
B 6 4 50,50;
B 4 8 57,31;
B 34 4 74,29;
B 4 6 69,13;
B 4 8 69,46;
B 6 4 88,16;
B 8 4 89,50;
B 4 4 89,29;
B 4 2 89,30;
B 12 4 95,29;
B 4 34 103,33;
B 4 4 103,50;
B 2 4 104,18;
B 10 8 120,36;
B 8 14 121,43;
B 12 8 123,18;
L CS;
B 4 2 4,35;
B 42 4 23,36;
B 4 62 45,33;
B 0 4 47,15;
B 0 4 47,50;
B 4 26 69,29;
B 2 0 68,42;
B 4 36 91,32;
B 8 62 113,33;
B 8 4 115,34;
B 8 4 129,16;
B 8 62 131,33;
L CD;
B 44 8 22,10;
B 94 14 47,53;
B 4 0 4,50;
B 4 0 4,10;
B 4 0 28,8;
B 4 0 28,50;
B 8 10 44,11;
B 12 8 46,16;
B 8 4 52,14;
B 46 8 71,16;
B 0 4 53,15;
B 0 4 53,50;
B 4 0 69,10;
B 4 0 69,50;
B 4 2 85,49;
B 0 4 85,16;
B 16 8 109,18;
```

```
B 24 14 113,53;
B 4 2 105,49;
B 0 4 105,18;
B 8 0 121,18;
B 4 0 121,50;
L CP;
B 6 8 7,24;
B 4 62 12,33;
B 4 6 22,39;
B 16 4 28,42;
B 4 4 22,20;
B 16 4 28,18;
B 4 24 36,52;
B 4 18 36,11;
B 4 4 37,29;
B 10 4 40,27;
B 4 4 45,27;
B 16 4 51,25;
B 4 4 57,35;
B 4 8 59,37;
B 4 4 59,41;
B 4 18 61,17;
B 8 4 63,26;
B 4 24 61,51;
B 4 10 67,29;
B 12 4 71,34;
B 4 32 77,48;
B 4 18 77,17;
B 14 4 82,26;
B 32 4 95,42;
B 4 0 89,31;
B 10 10 115,36;
B 4 54 113,37;
L CM;
B 8 8 4,34;
B 8 8 47,15;
B 8 8 47,50;
B 8 8 69,16;
B 8 8 69,42;
B 8 8 91,16;
B 8 8 91,50;
B 8 8 115,36;
B 8 8 129,18;
L CS;
B 8 8 4,34;
B 8 8 47,15;
B 8 8 47,50;
B 8 8 69,16;
B 8 8 69,42;
B 8 8 91,16;
B 8 8 91,50;
```

```
B 8 8 115,36;
B 8 8 129,18;
L CC;
B 4 4 4,34;
B 4 4 47,15;
B 4 4 47,50;
B 4 4 69,16;
B 4 4 69,42;
B 4 4 91,16;
B 4 4 91,50;
B 4 4 115,36;
B 4 4 129,18;
L CM;
B 8 8 4,50;
B 8 8 4,10;
B 8 8 28,8;
B 8 8 28,50;
B 8 8 53,15;
B 8 8 53,50;
B 8 8 69,10;
B 8 8 69,50;
B 8 8 85,16;
B 8 8 85,50;
B 8 8 105,18;
B 8 8 105,50;
B 8 8 121,50;
B 8 8 121,18;
L CD;
B 8 8 4,50;
B 8 8 4,10;
B 8 8 28,8;
B 8 8 28,50;
B 8 8 53,15;
B 8 8 53,50;
B 8 8 69,10;
B 8 8 69,50;
B 8 8 85,16;
B 8 8 85,50;
B 8 8 105,18;
B 8 8 105,50;
B 8 8 121,50;
B 8 8 121,18;
L CG;
B 4 4 4,50;
B 4 4 4,10;
B 4 4 28,8;
B 4 4 28,50;
B 4 4 53,15;
B 4 4 53,50;
B 4 4 69,10;
B 4 4 69,50;
```

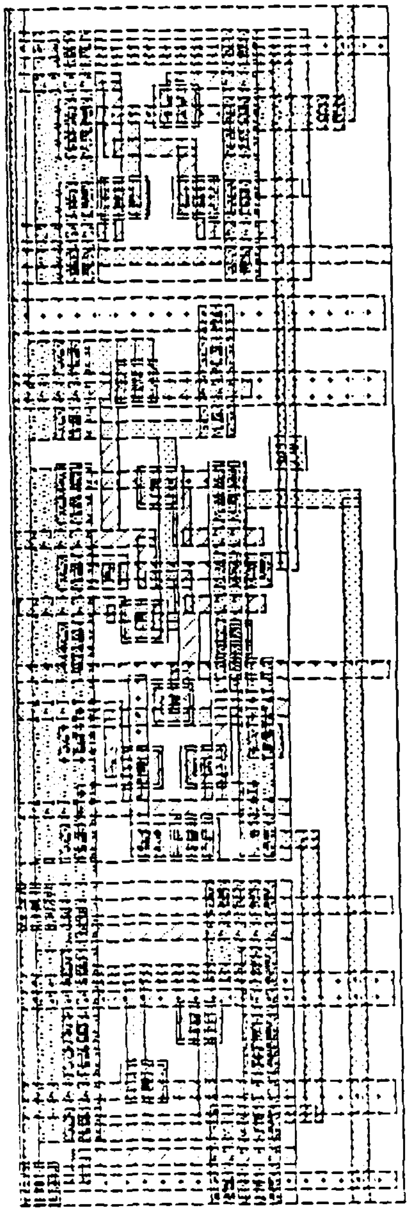
```
B 4 4 85,16;
B 4 4 85,50;
B 4 4 105,18;
B 4 4 105,50;
B 4 4 121,50;
B 4 4 121,18;
L CM;
B 8 8 4,24;
B 8 8 22,36;
B 8 8 22,22;
B 8 8 37,29;
B 8 8 57,35;
B 8 8 89,31;
B 8 8 89,31;
L CP;
B 10 10 4,24;
B 10 10 22,36;
B 10 10 22,22;
B 10 10 37,29;
B 10 10 57,35;
B 10 10 89,31;
B 10 10 89,31;
L CG;
B 4 4 4,24;
B 4 4 22,36;
B 4 4 22,22;
B 4 4 37,29;
B 4 4 57,35;
B 4 4 89,31;
B 4 4 89,31;
(END WIRE DEFINITIONS);

(START VIA DEFINITIONS);
(END VIA DEFINITIONS);

(START CALL STATEMENTS);
(END CALL STATEMENTS);

(START ITER STATEMENTS);
(END ITER STATEMENTS);
(bounds SUB 0,2 135,64);
DF;
C 4888 T 0,-2;
E
```

A>mcifplot* of SUB.CIF
Window: -3190 -98 -50 7100
Scale: 1 micron is .1118881 inches



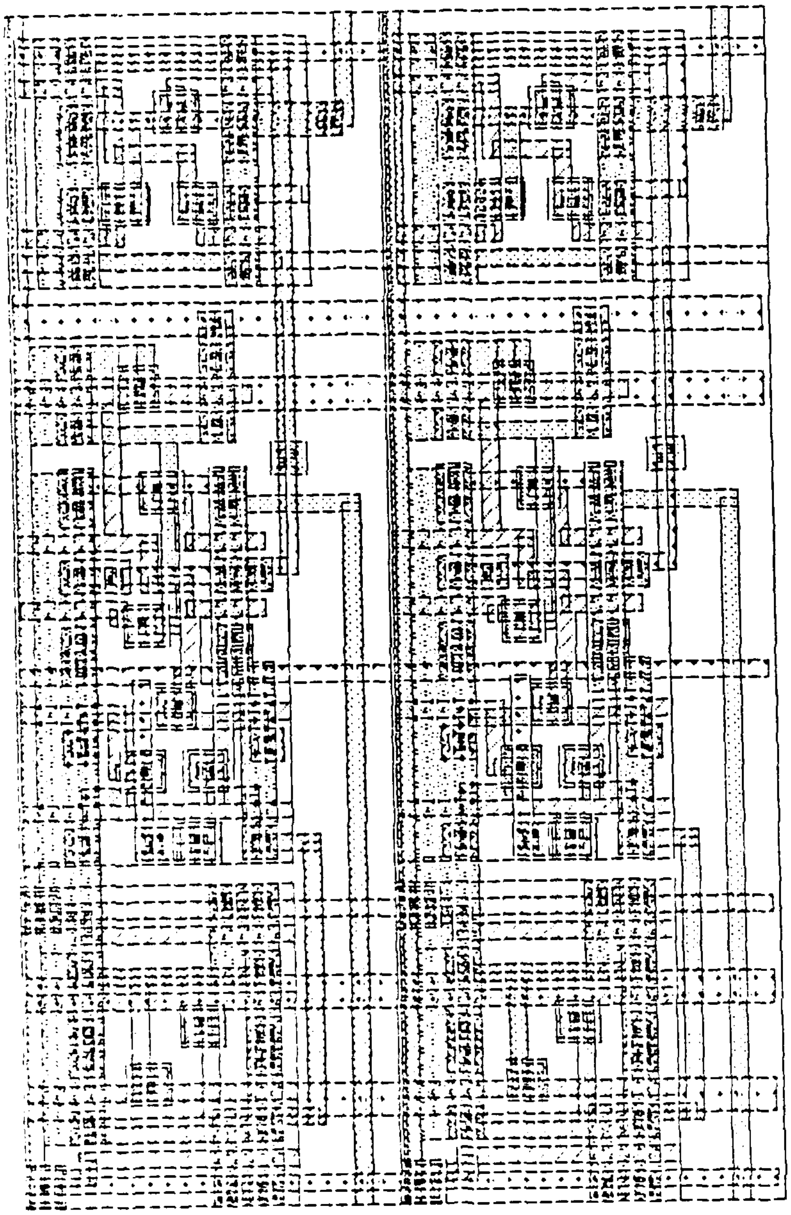
```
DS 19942 50 1;
(TITLE BINBOUT 19942);

(START PORT DEFINITIONS);
(END PORT DEFINITIONS);

(START RECTANGLE DEFINITIONS);
(END RECTANGLE DEFINITIONS);

(START WIRE DEFINITIONS);
(Wire 0,26 F W4 f170 u24 );
(Wire 6,40 S W4 d22 );
(Wire 22,44 F W4 d8 f66 u8 );
(Wire 26,40 S W8 d22 );
(Wire 52,40 S W8 d22 );
(Wire 72,40 S W4 d22 );
(Wire 77,100 F W4 f212 );
(Wire 84,91 D W14 b22 );
(Wire 129,40 S W4 d22 );
(Wire 155,44 S W4 d4 f26 F f96 u6 );
(Wire 197,40 S W8 d22 );
(Wire 215,40 S W8 d22 );
(Wire 229,40 S W4 d24 );
(Wire 263,48 D W8 d18 F W4 d4 f26 );
(Wire 279,34 S W4 d18 );
L CM;
B 170 4 85,26;
B 4 8 22,40;
B 68 4 54,36;
B 212 4 183,100;
B 4 10 88,39;
B 4 26 170,37;
B 96 4 229,40;
B 4 4 263,28;
B 28 4 275,26;
B 4 8 277,42;
L CM2;
B 4 22 6,29;
B 8 22 26,29;
B 8 22 52,29;
B 4 22 72,29;
B 4 22 129,29;
B 4 4 155,42;
B 28 4 167,40;
B 8 22 197,29;
B 8 22 215,29;
B 4 24 229,28;
B 4 18 279,25;
L CD;
B 22 14 73,91;
B 8 18 263,39;
```

```
L CM;
B 8 8 181,40;
L CM2;
B 8 8 181,40;
L CC;
B 4 4 181,40;
L CM;
B 8 8 263,30;
L CD;
B 8 8 263,30;
L CV;
B 4 4 263,30;
(END WIRE DEFINITIONS);

(START VIA DEFINITIONS);
(END VIA DEFINITIONS);

(START CALL STATEMENTS);
(Call MEM 0,40 bounds 0,0 78,64);
C 4139 T 0,40;
(Call SUB 84,40 bounds 0,2 135,64);
C 4888 T 84,38;
(Call MASK 223,34 bounds 0,0 60,66);
C 32077 T 223,34;
(END CALL STATEMENTS);

(START ITER STATEMENTS);
(END ITER STATEMENTS);
(bounds BINBOUT 0,16 289,104);
DF;
C 19942 T 0,-16;
E
```



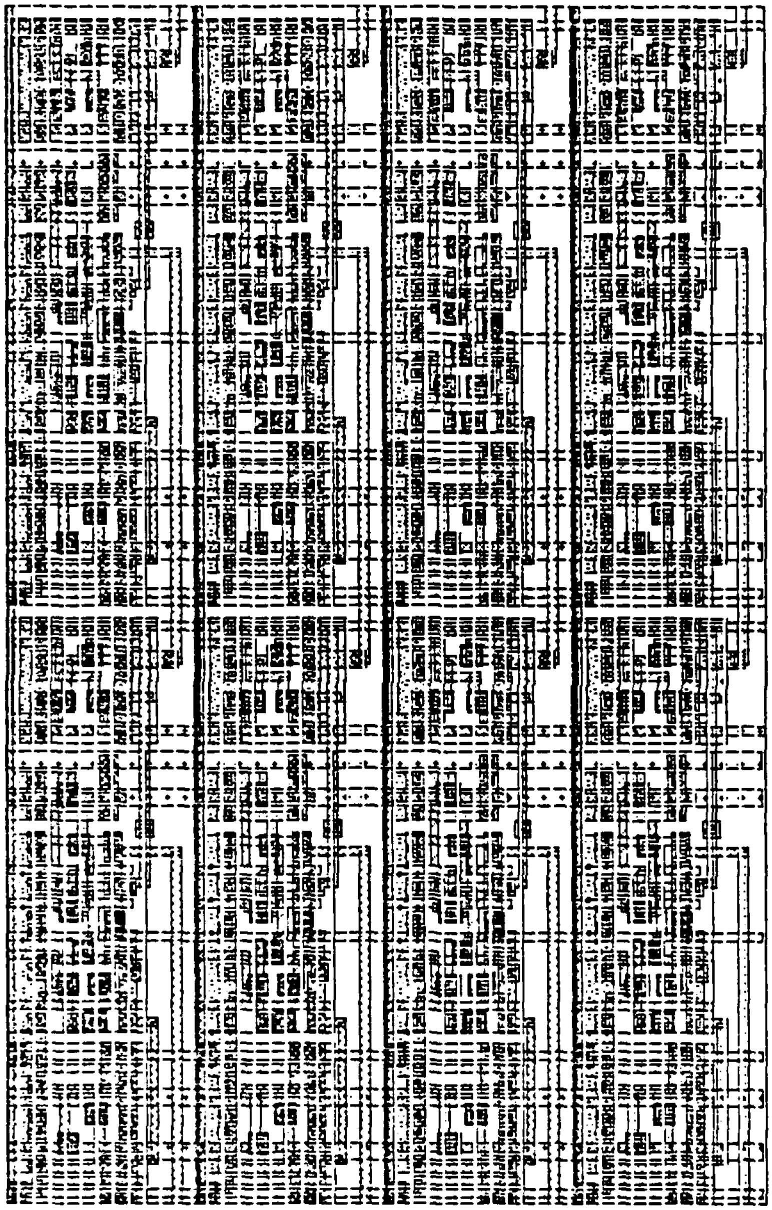
MCIFPLOT of BINBOUT.CIF

```
DS 32392 50 1;
(TITLE MEM2 32392);

(START PORT DEFINITIONS);
(END PORT DEFINITIONS);

(START RECTANGLE DEFINITIONS);
(END RECTANGLE DEFINITIONS);

(START WIRE DEFINITIONS);
(END WIRE DEFINITIONS);

(START VIA DEFINITIONS);
(END VIA DEFINITIONS);

(START CALL STATEMENTS);
(END CALL STATEMENTS);

(START ITER STATEMENTS);
(Iter BINBOUT 1,2 289,85 0,0  bounds 0,0 0,0);
C 19942 T 0,0;
C 19942 T 0,85;
(END ITER STATEMENTS);
(bounds MEM2 0,0 0,85);
DF;
C 32392 T 0,0;
E
```



MCIFPLOT of MEM2.CIF

```
DS 32398 50 1;
(TITLE MEM8 32398);

(START PORT DEFINITIONS);
(END PORT DEFINITIONS);

(START RECTANGLE DEFINITIONS);
(END RECTANGLE DEFINITIONS);

(START WIRE DEFINITIONS);
(END WIRE DEFINITIONS);

(START VIA DEFINITIONS);
(END VIA DEFINITIONS);

(START CALL STATEMENTS);
(END CALL STATEMENTS);

(START ITER STATEMENTS);
(Iter BINBOUT 2,4 289,85 0,0  bounds 0,0 0,0);
C 19942 T 0,0;
C 19942 T 0,85;
C 19942 T 0,170;
C 19942 T 0,255;
C 19942 T 289,0;
C 19942 T 289,85;
C 19942 T 289,170;
C 19942 T 289,255;
(END ITER STATEMENTS);
(bounds MEM8 0,0 289,255);
DF;
C 32398 T 0,0;
E
```



MCIFPLOT of MEM8.CIF

Bibliography

1. Ahl, David H. "The First Decade of Personnal Computting," Creative Computing 10:30-45 (November 1984).

2. Ahl, David H. "Ascent of the Personnal Computer," Creative Computing 10:80-82 (November 1984).

3. Chakravarty, Dev. "The Incredible Shrinking Circuits," Computers and Electronics, 22:64-67+ (April 1984).

4. Feuer, Michael. "VLSI Design Automation: An Introduction," Proceedings of the IEEE, Vol 71, No 1:5-9, (January 1983).

5. Gladstone, Bruce E. "PCs and Mainframes Automate Electronic Design," IEEE Computer Conference Spring 84, 384-387, IEEE Computer Society Press, Los Alamitos, CA, (February 1984).

6. Gray, Stephen B. "The Early Days of Personnal Computers," Creative Computing 10:6-14 (November 1984).

7. Horton, Kirk, "A Microcomputer-based program for printing Check Plots of Integrated Circuits Specified in Caltech Intermediate Form, Air Force Institute of Technology, December 1984.

8. Israel, M. and G. Noquez. "EMILIE2: A Microprocessor Based Interactive Graphic Editor for Integrated Circuit Design," Proceedings of the IEEE 3rd Annual Workshop on Interactive Computing: CAD/CAM: Electrical Engineering Education, 69-71, IEEE Computer Society Press, Los Alamitos CA, (October 1984).

9. Jadrnicek, Rik "Computer Aided Design," BYTE, 9:172-09+ (January 1984).

10. Kernigan, B. W. and D. M. Ritchie, The C Programming Language, Englewood Cliffs, New Jersey: Prentice-Hall Inc. 1978.

11. Lee, Alex, "Engineering Design on Micros," Computers and Electronics, 86-90+ (September 1984).

12. Mead, Carver and Lynn Conway, Introduction to VLSI Systems, Reading, Massachusetts: Addison-Wesley Publishing Company, 1980.

13. Mims III, Forrest M. "The Altair Story" Creative Computing," 10: 17-27 (November 1984).

14. Rubenstein, Charles, "Computer Aided," Computers and Electronics, 67-77 (May 1983).

15. Saxe, Tim and others, "CLL - A Chip Layout Language (version 4) White Paper, Air Force Institute of Technology, December 1984.

16. Smith, Kent, International Symposium on Circuits and Systems, 505-508, 1983.

17. Sussman-Fort, S. E. and others, "Line Printer Plotting Programs for VLSI," Proceedings of the IEEE 1st Annual Workshop on Interactive Computing: CAD/CAM: Electrical Engineering Education, 97-100, IEEE Computer Society Press, Los Alamitos, CA (October 1982).

18. Thompson, Andrew V. "MICRO-CAP: An Analog Circuit Design System for Personnal Computers," IEEE Computer Conference Spring 84, 388-390, IEEE Computer Society Press, Los Alamitos, CA, (February 1984).

19. Thompson, Andrew V. "CAE Tools for Personnal Computers," IEEE 1984 International Symposium on Circuits and Systems, 152-155, Institute of Electrical and Electronics Engineers.

VITA

Captain Steven C. Morrese was born on 3 February 1958 in Kenmore, New York. He graduated from high school in O'Fallon, Illinois, in 1976 and attended the Southern Illinois University at Edwardsville, Illinois, from which he received the degree of Bachelor of Science in Mathematics in June 1980. Upon graduation he was commissioned in the USAF through the ROTC program. He received training at Keesler AFB, MS, from Oct 1980 until Jan 1981 and was then assigned to Hanscom AFB, MA, as a Computer Engineer. His military career was enhanced by being selected to attend the Air Force Institute of Technology to pursue a graduate degree in Computer Science starting in June 1984.

Permanent Address: 115 Ruth Drive
O'Fallon, Illinois 62269

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

1b. RESTRICTIVE MARKINGS

2a. SECURITY CLASSIFICATION AUTHORITY

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE

3. DISTRIBUTION/AVAILABILITY OF REPORT
Approved for public release;
Distribution unlimited

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
AFIT/GCS/ENG/85D-11

5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION
School of Engineering

6b. OFFICE SYMBOL *(If applicable)*
AFIT/ENG

7a. NAME OF MONITORING ORGANIZATION

6c. ADDRESS *(City, State and ZIP Code)*
Air Force Institute of Technology
Wright-Patterson Air Force Base, OH 45433

7b. ADDRESS *(City, State and ZIP Code)*

8a. NAME OF FUNDING/SPONSORING ORGANIZATION
Defense Advanced Projects

8b. OFFICE SYMBOL *(If applicable)*
DARPA/IPTO

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

8c. ADDRESS *(City, State and ZIP Code)*
1400 Wilson Blvd.
Arlington, VA 22209

10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.

11. TITLE *(Include Security Classification)*
See Box 19

12. PERSONAL AUTHOR(S)
Steven C. Morrese, Capt, USAF

13a. TYPE OF REPORT
MS Thesis

13b. TIME COVERED
FROM ________ TO ________

14. DATE OF REPORT *(Yr., Mo., Day)*
1985 December

15. PAGE COUNT
320

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB. GR.
09	02	

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
Microcomputers; Computer Aided Design;
Computer Programs; Integrated Circuits

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

Title: A Microcomputer-Base Menu-Driven Chip Layout Language

Thesis Advisor: Harold W. Carter, Lt Col, USAF

Approved for public release: IAW AFR 190-1/.
Lynn Wolaver 16 JAN 86
LYNN E. WOLAVER
Dean for Research and Professional Development
Air Force Institute of Technology (~~ATC~~)
Wright-Patterson AFB OH 45433

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT
UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. □ DTIC USERS □

21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL
Harold W. Carter, Lt Col, USAF

22b. TELEPHONE NUMBER *(Include Area Code)*
513-255-6913

22c. OFFICE SYMBOL
AFIT/ENG

EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

The Menu Driven Chip Layout Language (MDLL) is a microcomputer-based design tool used for representing the physical layout of integrated circuit designs in the Caltech Intermediate Form (CIF). MDCLL is based on the Chip Layout Language (CLL) created by Tim Saxe of Stanford University and as modified at the Air Force Institute of Technology. MDCLL leads the user through interactive menu-driven functions, allowing the user to represent a circuit design in CIF; however, the user sees a language similar to CLL even though the file is stored on disk as CIF. The output CIF file is suitable for use with other Computer Aided Design tools or for use in the fabrication process.

MDCLL uses a cell-oriented design methodology in which the user defines cell, and then the user combines these cells with the necessary interconnects to form the layout of an entire integrated circuit. Cell interconnections are possible by placing wires, abutting cells, or by overlapping cells.

The main functions of MDCLL are to create a new cell, delete a cell, place a cell, move a cell, modify a cell, interconnect cells, and print the CIF file of a cell.